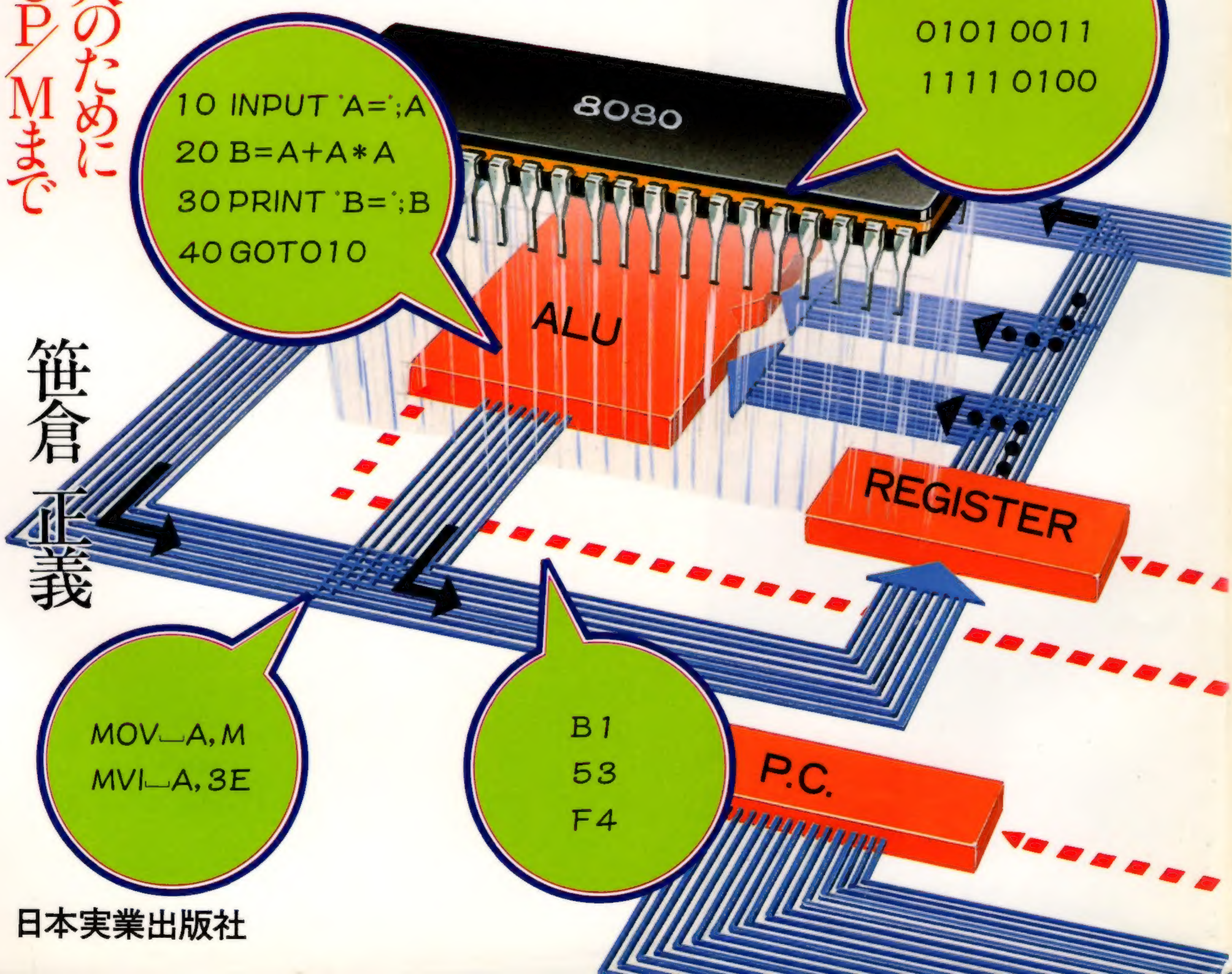


# 8080 マイコン C P U 大研究

絵と文字

● 8080、Z80をもつと知りたい人のために  
● BASICからアセンブラマシン語、C P / Mまで

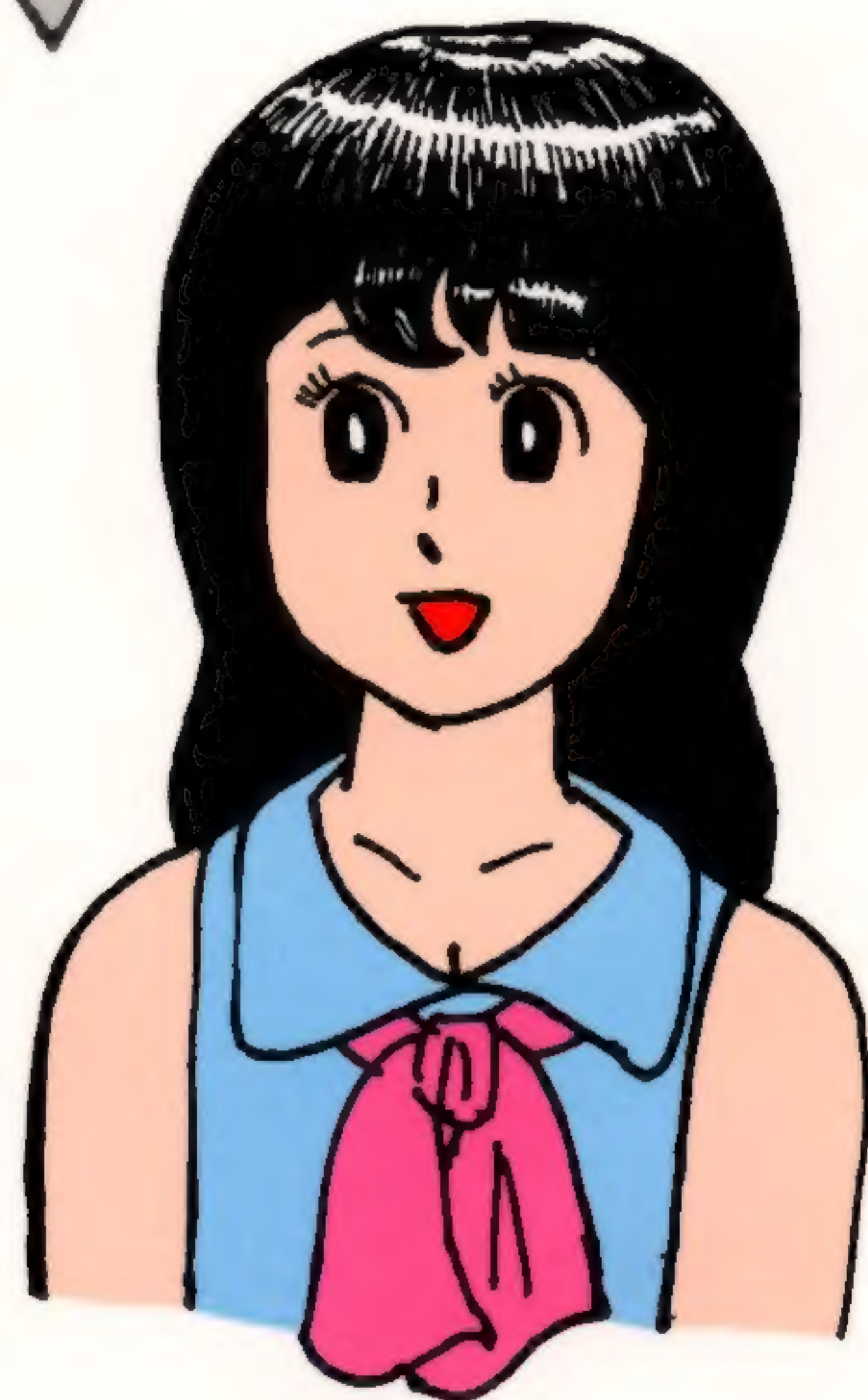
と6809





\*PC、MZ、MSXなど  
80系のマイコンユーザー  
のみなさん、ぜひこの本を  
お読みください。

\*FMなど68系のユーザーの  
みなさんにも、きっと  
お役に立ちます。



友子が絵で解説した  
この本を読んで  
いただければ……

- CPUのことがわかる
- マシン語のことがわかる
- マイコンの高度な活用がわかる



# 8080マイコン大研究

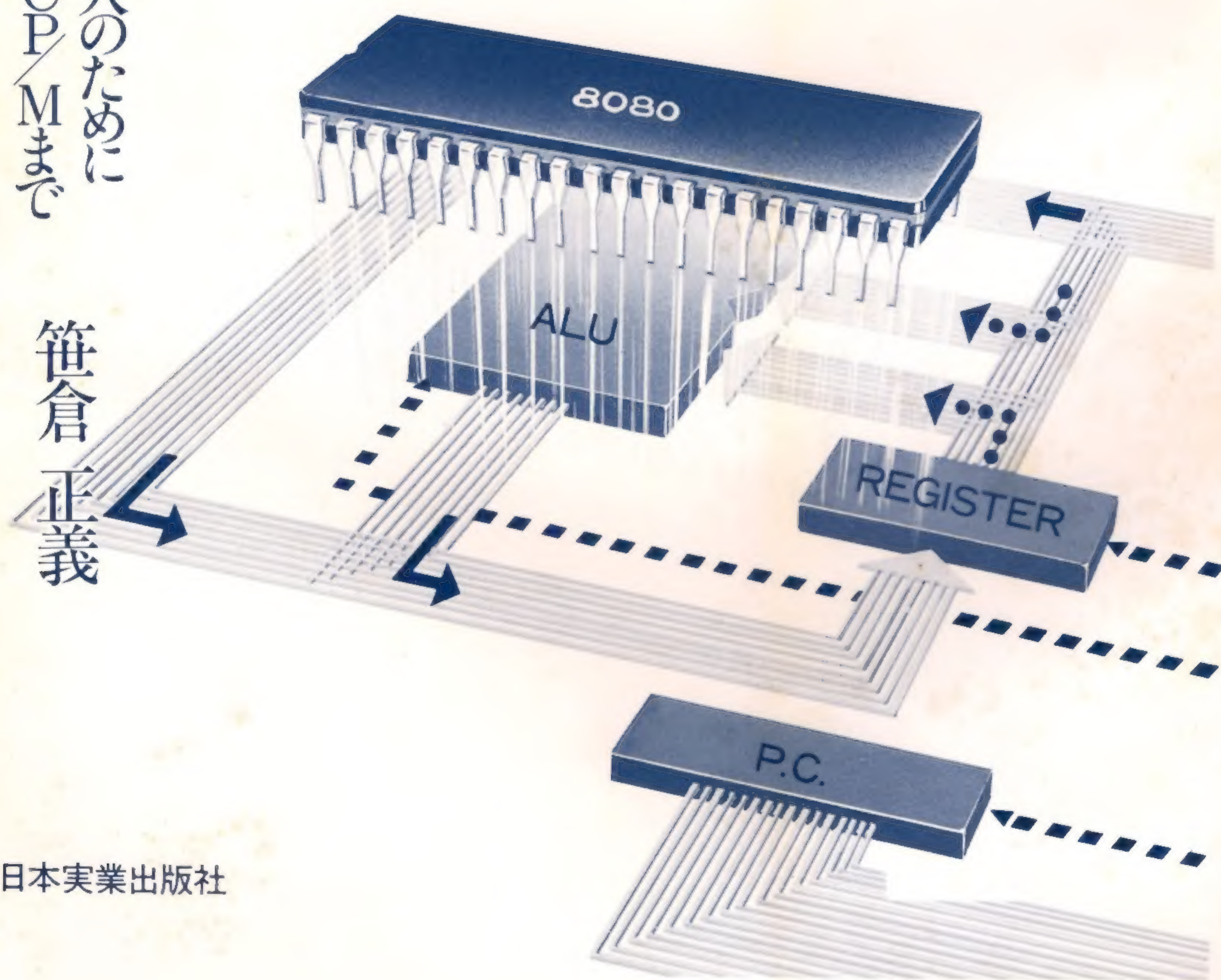
絵とまち

と6809

●8080、Z80をもつと知りたい人のために  
●BASICからアセンブラマシン語、CP/Mまで

笹倉正義

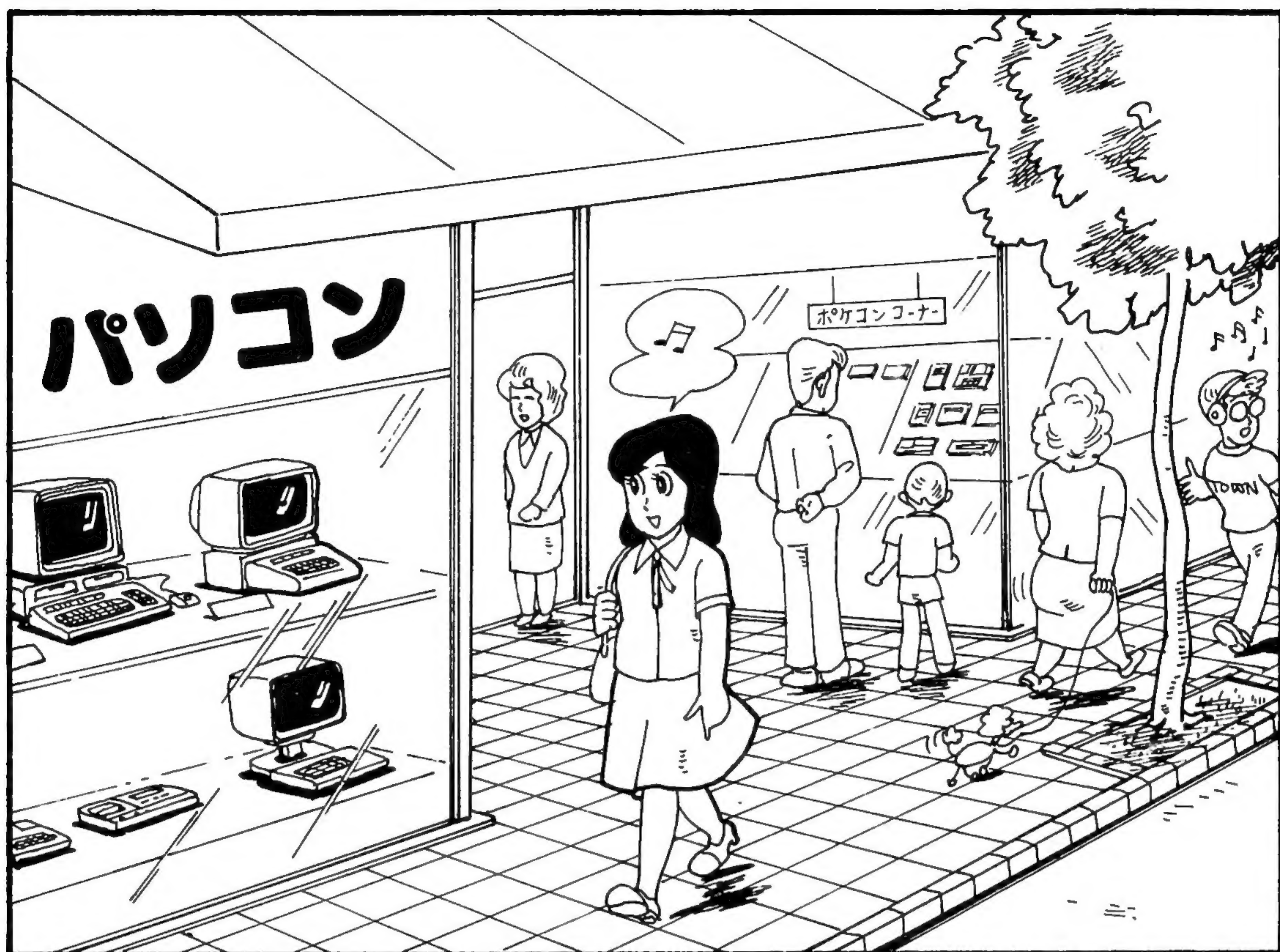
日本実業出版社



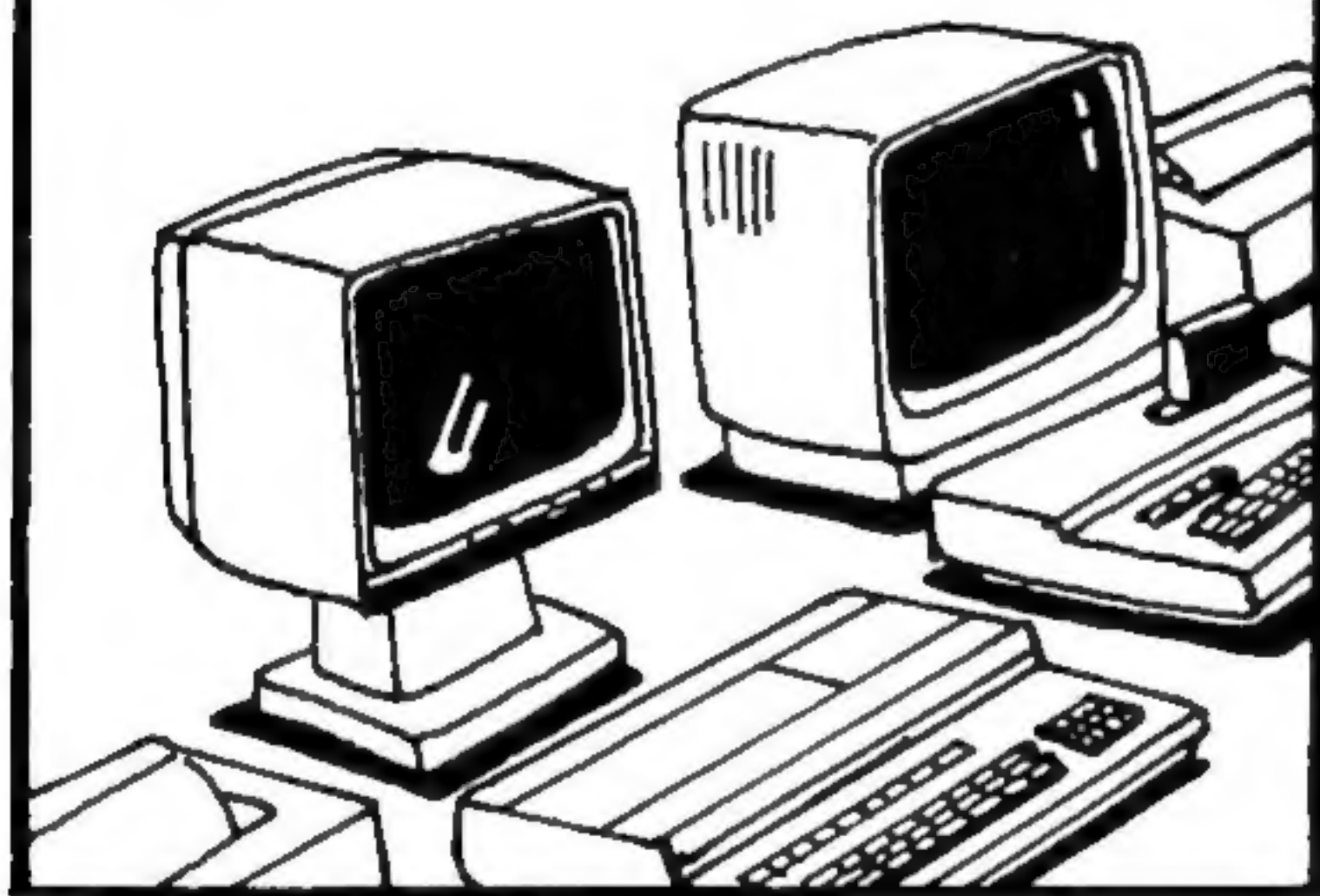








パソコンは、今や家電メーカーも進出、中学生や小学生まで持っている時代です。



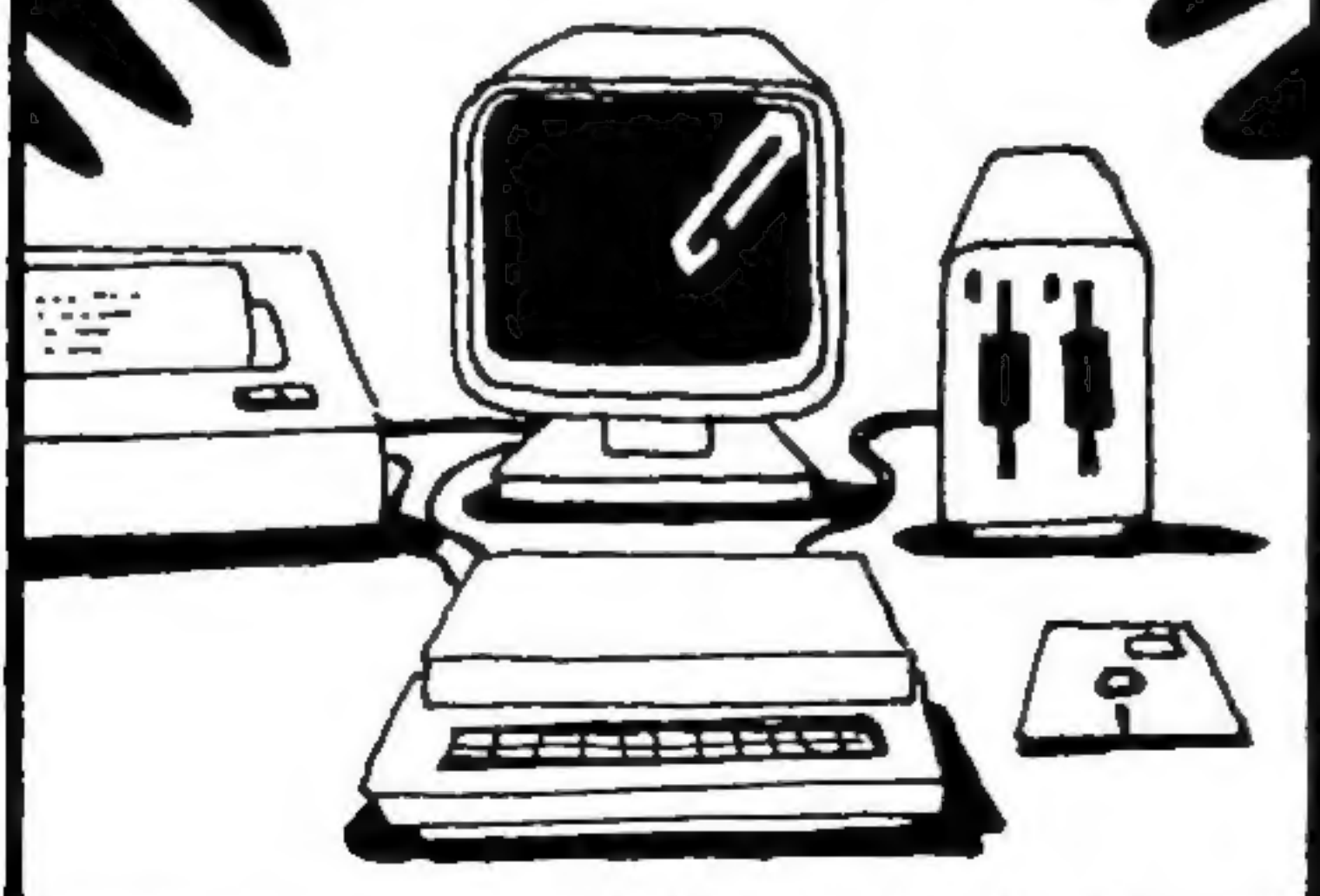
きょうはパソコンのウィンドショッピングです。



あ、はじめまして。私は友子です。どうぞよろしく。



パソコンといってもコンピュータなのですから、そう簡単に自在に使いこなすというわけにはいきません。



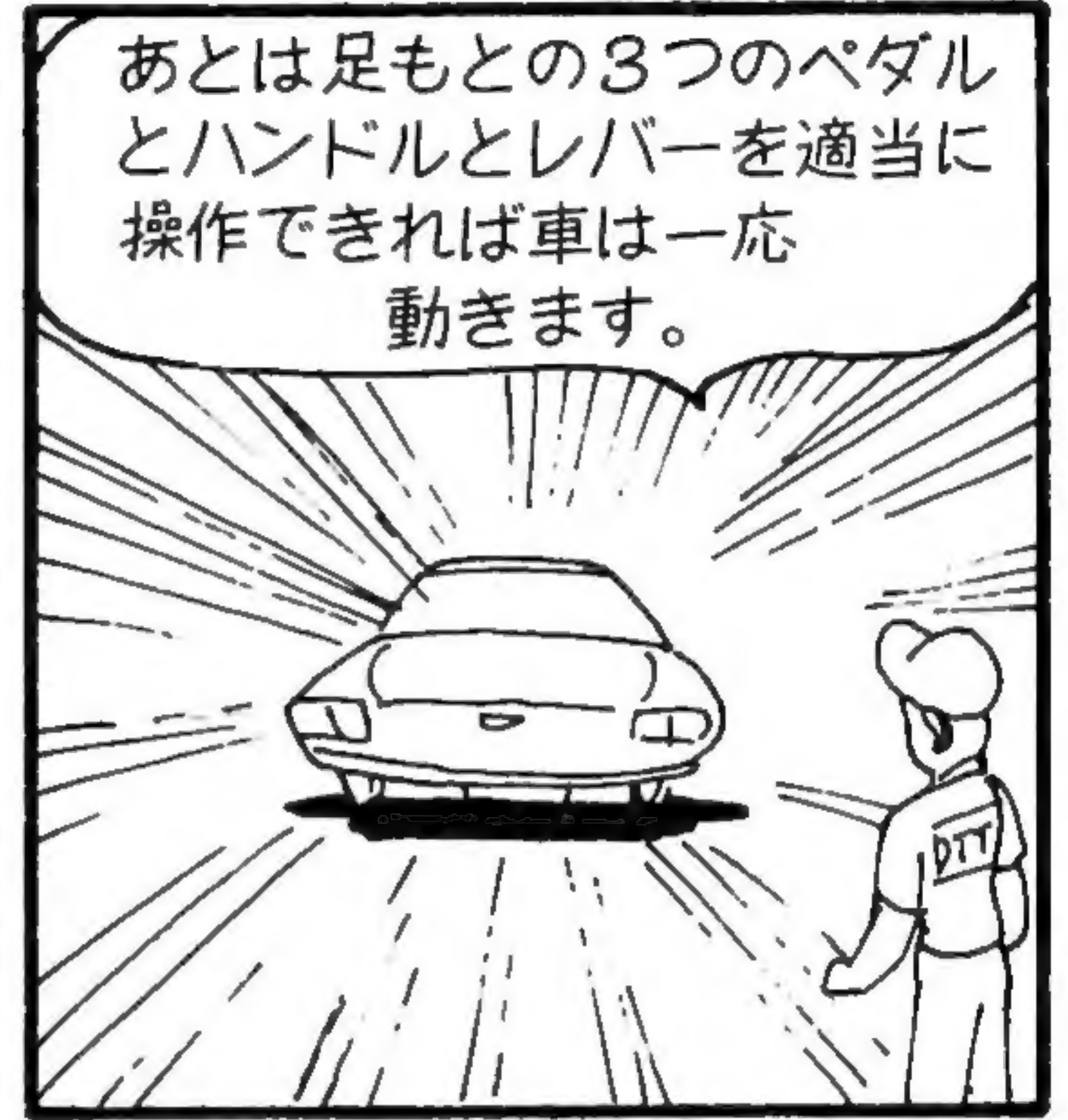
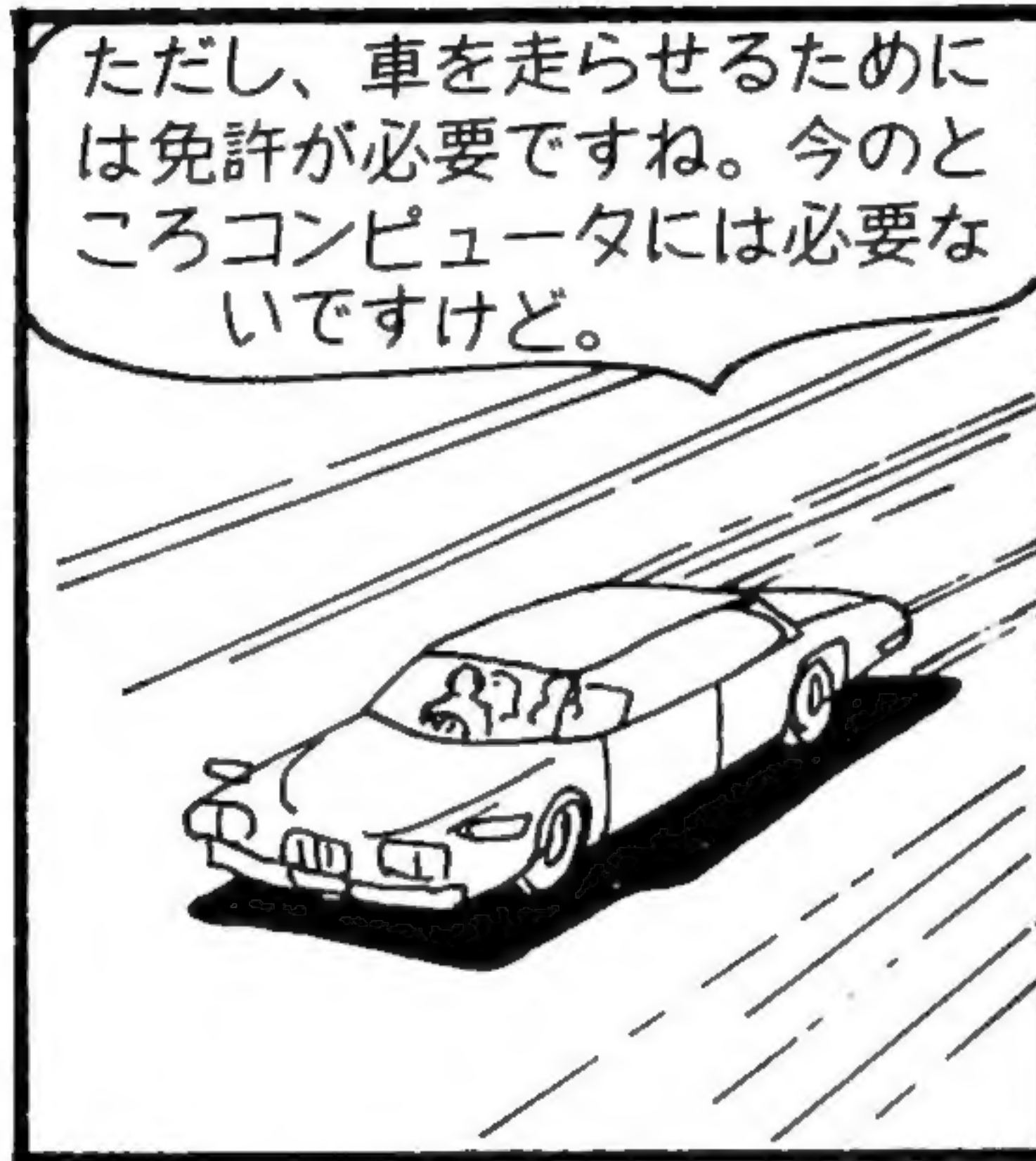
ことばだけがCMベースで先行し、中身がついていないようなのです。



ところが、ゲーム以外になにかできるのかというとなかなかみたいです。







マイコンの勉強は、まずBASICから入るのが普通です。BASICは人間の言葉に近い表現なので、比較的わかりやすいからです。しかし、BASICだけで使っているうちは、CPUのことはなかなかわかりません。

CPUがいったいどんな働きをどのようにしているのか知りたいと思いませんか。それがわかれば、マシン語がわかってきます。そして、コンピュータの活用の幅がぐーんと広がってくるはずですよ。



……というようなわけで、友子ちゃんがとり出しましたこの本は、マイコンの入門書としてはちよつと変わっています。一応はマンガの格好をしていますが、面白いマンガではありません。けれども、マイコンのことをもっと知りたい、という人にとっては、本当に役立つ内容を盛り込んであります。

言葉の数を多くして文章でくどくどと説明するのではなく、ポイントをできるだけ簡潔におさえるようにしています。そのため、若干のみ込みにくいところがあるかもしれませんが、あるいは、あなたがまだマイコンやその本にあまりなじみがない場合、一回ざっと読んでいただけではわかりにくいでしょう。ハードに興味のない人たちにとっては、若干退屈なところも出てきます。

しかし、それで放り出さずに、何回か、じっくりと、繰り返し読んでみてください。眺めてみてください。あきたら、しばらく放っておいて、またしばらくして手にとってみましょう。

どうしても読めないところは、どんどんとばしていただく。でも何回かめくっているうちに、だんだんと、わかってきます。わかる箇所が少しずつ増えてきます。これは、そういう本なのです。

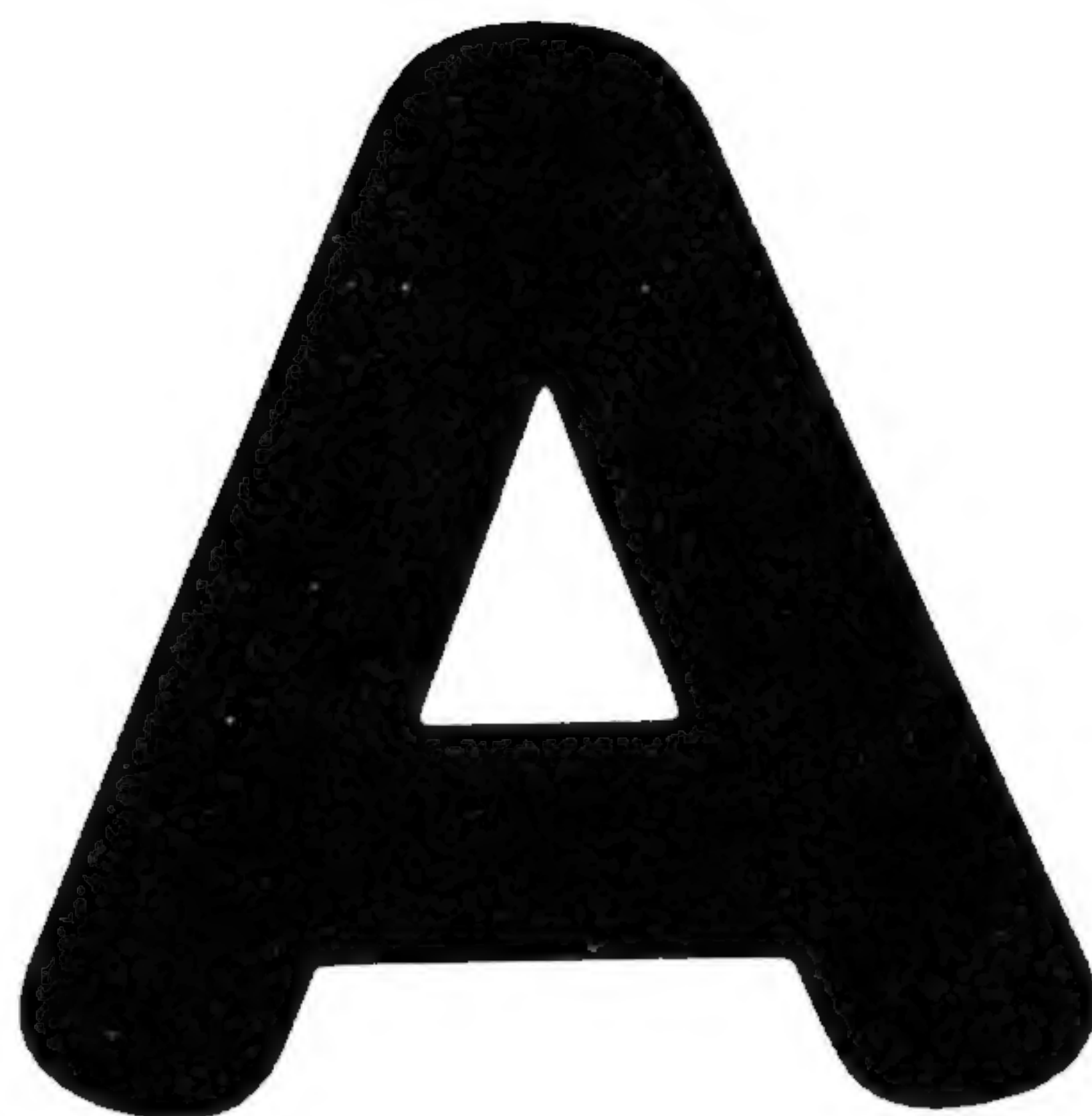
\*

\*

さて、かめばかむほど味の出るスルメのようなこの本の内容ですが……



# もくじ



## BASICはパソコン理解の第一歩

1	パソコンはBASICが使える	12
2	ROM型パソコンとオールRAM型パソコン	15
3	ループを使って累加計算してみよう	19
4	FOR〜NEXT文を使いこなそう	25
5	配列がわかればメモリもわかる	28
6	フロッピーにデータファイルをつくる	40

まず、定石どおりBASICのお勉強から……。

「BASICなんて勉強してみたってしょうがない。

これからは簡易言語だ」などという人が多くなっています。それも一理あります。要は、使う人のニーズの問題でしかないでしょう。

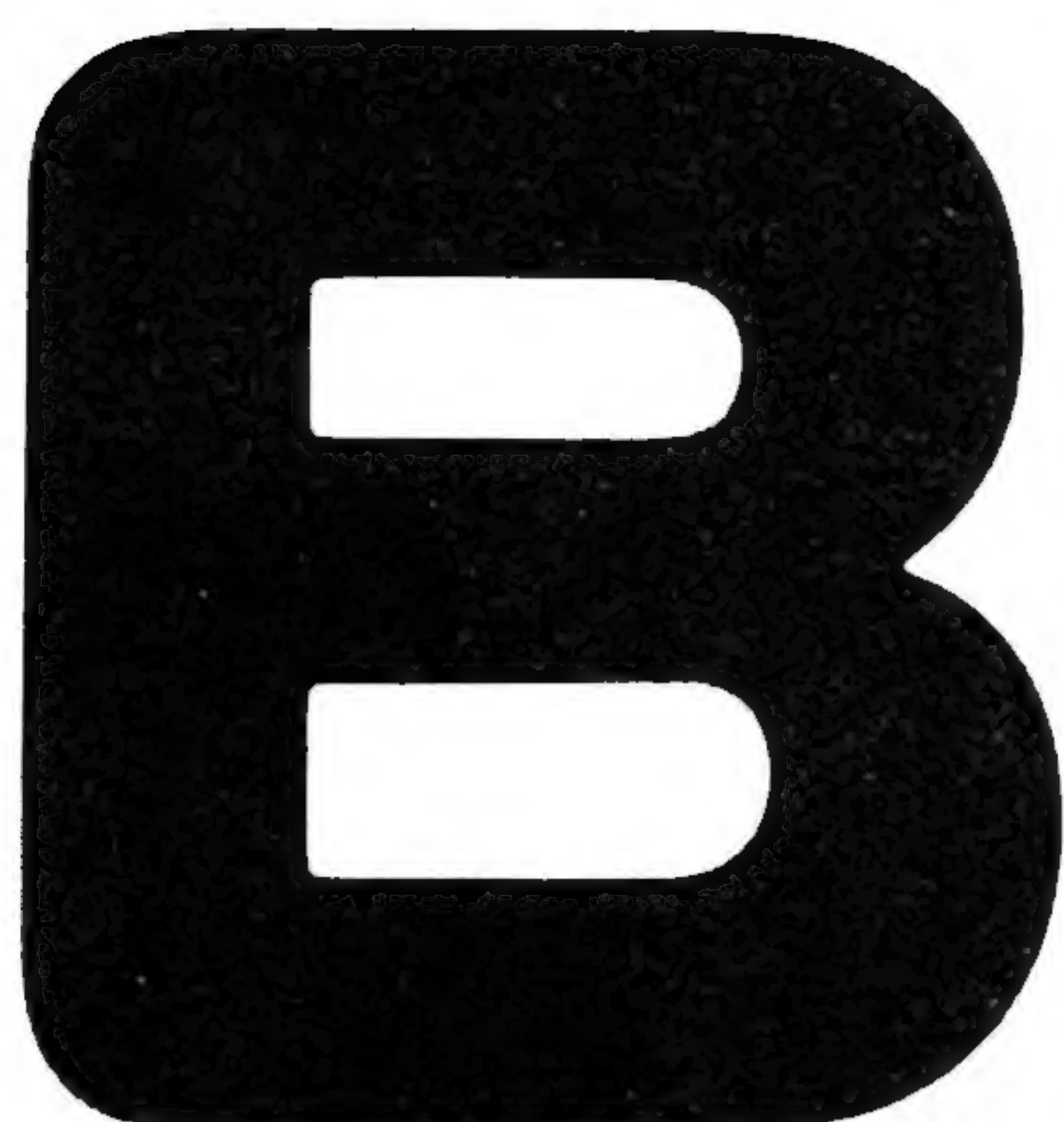
BASICも、このA章でやっている程度のことすら理解せず、やろうともせず、そういう論に追随するのはどんなものでしょうか。

ここでは、パソコンを理解する第一歩は、やはりBASICからということで、簡単なBASICの実用知識を整理し、データファイルの操作ができるところまで述べています。

これが完全にわかれば、あとはあなた自身のニーズに応じて、バツとヒラメクものが何かあるはずです。



# マシン語がわかればもっと使いこなせる



1	マシン語との出会い	54
2	ワンボードマイコンとモニタ	55
3	マシン語の「いろは」は0か1 <small>(ロー) (ハイ)</small>	60
4	CPUのしくみ(8080の場合)	66
5	マシン語のしくみ	68
6	8080コード表	73
7	マシン語こんな命令もある	74
8	マシン語でプログラムだ!	80
9	プログラムカウンタとスタックポインタ	92
10	Z80	100
11	Z80コード表	106
12	6502と6809	112

最近の傾向としては、確かに簡易言語派が増えてはいます。けれど、その一方で、「BASICは一応卒業したから、今度はマシン語に挑戦したい」という人々もたくさんいるはずですよ。

別にマニアというわけではないけれど、同じやるならもう少し突っ込んでみたい、という自然な願望です。

現に、ゲームなどはマシン語を使わないと、あまり面白いものはありません。マイコン誌などに載るプログラムリストも、16進表記のマシン語リストの方が多くなっています。ただわけもわからず打ち込んでもつまらないですね。

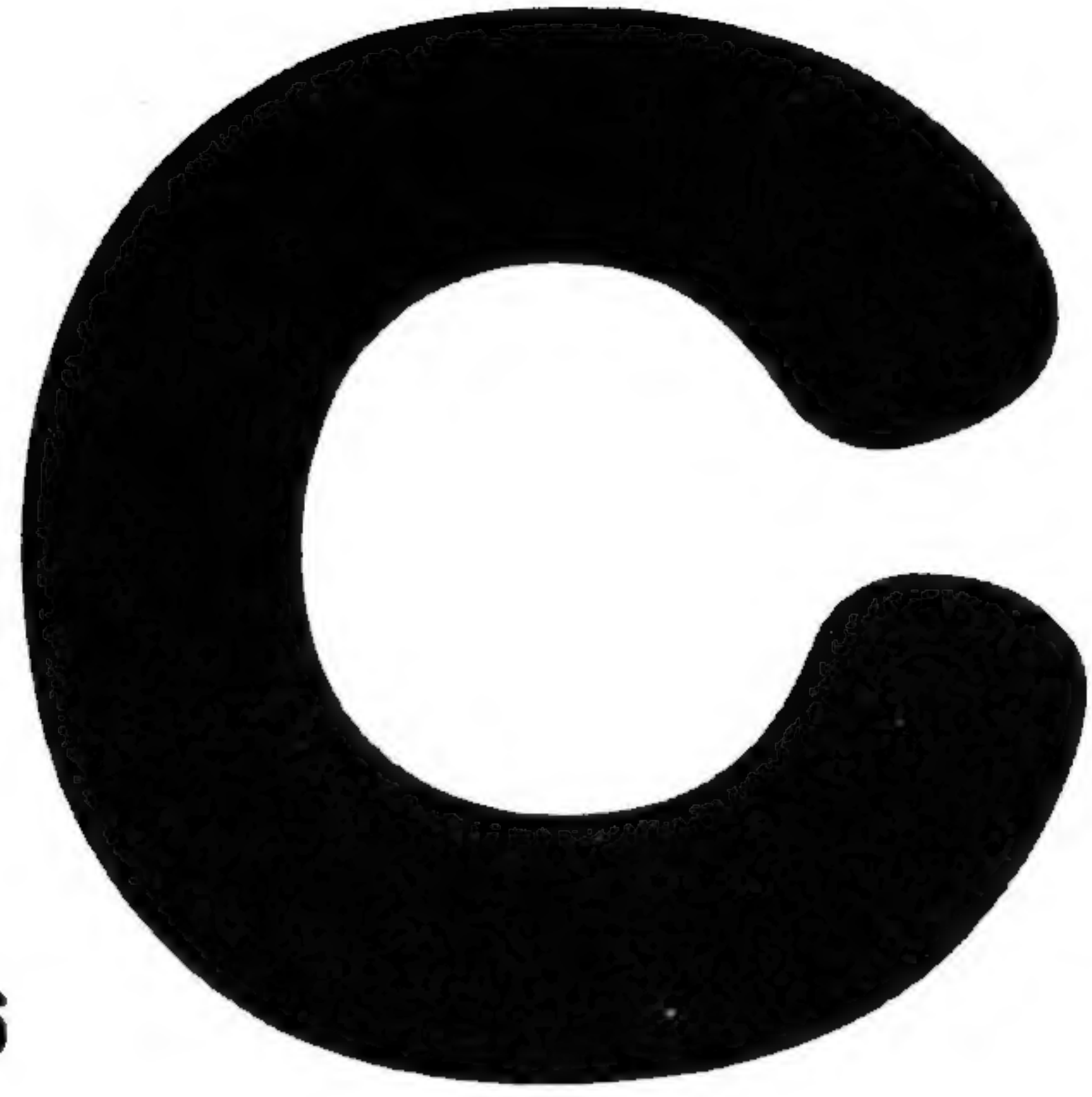
少しでもマシン語がわかれば、もっともって使い方が広がってきます。楽しさが増えます。

少々めんどくさいのですが、ほら、例の「知るは喜びなり」と申します」という名文句もあるではありませんか。がんばりましょう。

マシン語はCPUによって異なります。ここでは8080を中心に説明します。それは、8080がわかれば80系のイシは全部わかり、Z80もわかるからです。おまけで6809についてもちよっとふれました。



# CPUの中はどんなになっているか



1	石（IC）の大きさ	120
2	ロジック用IC	121
3	フリップフロップ	125
4	ワンショットパルサー	129
5	ROMとRAM	131
6	メモリ容量とチップセレクト	133
7	トライステート・ゲート	137
8	CPUと制御信号	139
9	CPUとクロック	142
10	ハード的なオペコード（割込命令）	154
11	入出力装置用ポート（I/Oポート）	160

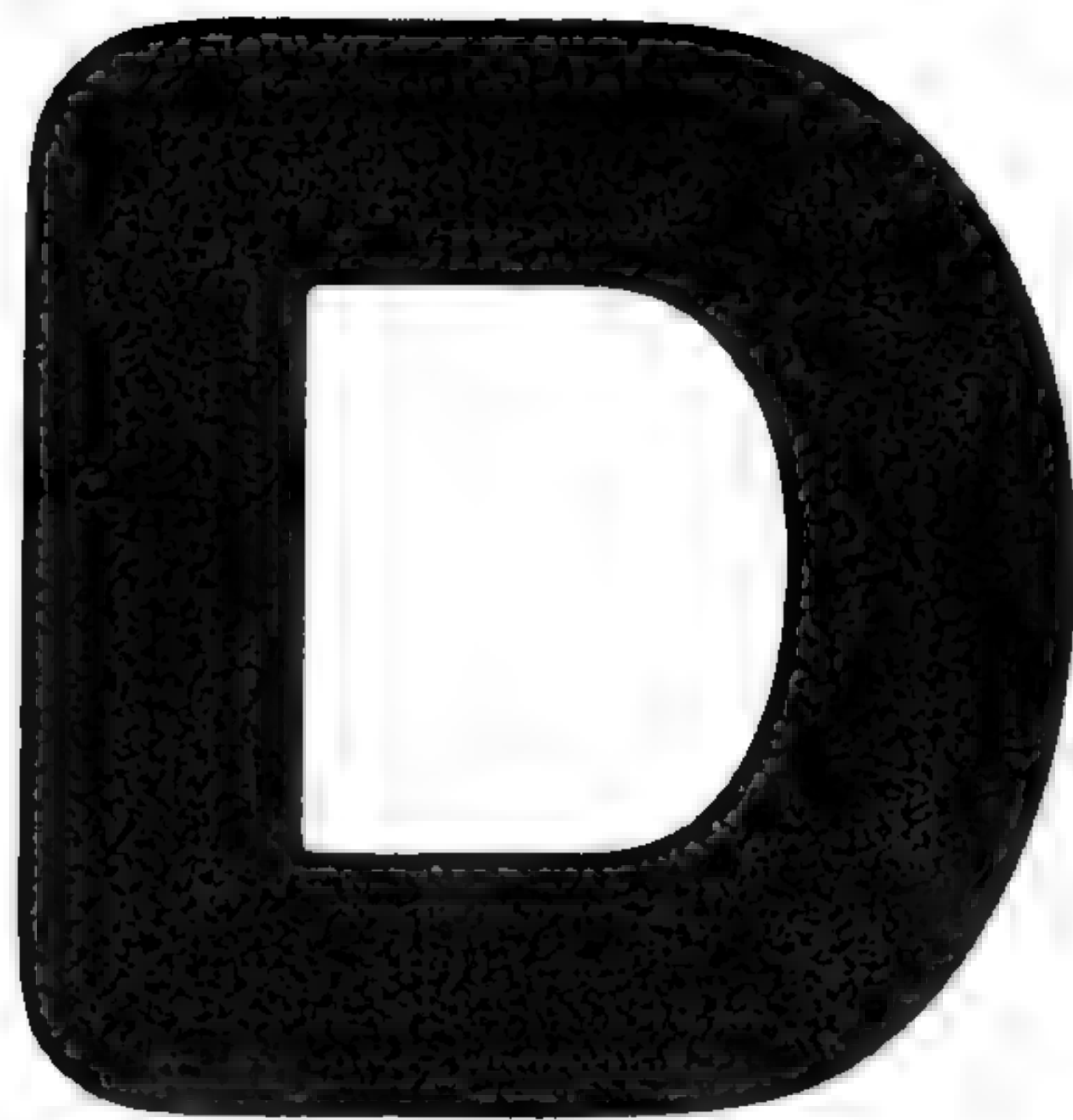
マシン語をやるなら、何かイシを決めてやらないと、何が何だかわからなくなる、といいます。それは、自分がどのパソコンを持っているか、で決まります。でも、マシン語とはどんなものがわかれば、あとは比較的早いです。6809の人も、これですと入門が楽になりますよ。

それにしても、CPUというのはちっちゃなものなのに、すごいことができるものです。マシン語をよりよく知るためのCPUの中の話を少々……。

このあたり、回路がどうだとか、クロックがどうだとかいう話になり、ちよつとむずかしいかもしれません。まア、だいたいの感じを理解していただければ……。



## CP/M アセンブラの活用知識



1	CP/Mとは何か	176
2	8080アセンブラを走らせよう	178
3	Z80アセンブラを試す	190

CPUの機能をフルに活かすためのソフトの中心になるのが、OS(オペレーティングシステム)といわれるものです。デジタルリサーチ社が製品化しているCP/Mとは、このOSの一種なのです。A社のパソコンにはすでにそれ用のOSがあるのですが、そのうえ、さらに標準OSが必要になるというのはどういことでしょうか。

CP/Mは、本来的には、強力なアセンブラなどを備えています。そしてZ80上で動きます。ですからZ80コンパチマシン(数が多い)のソフト開発がその主要な役割なのですが……。

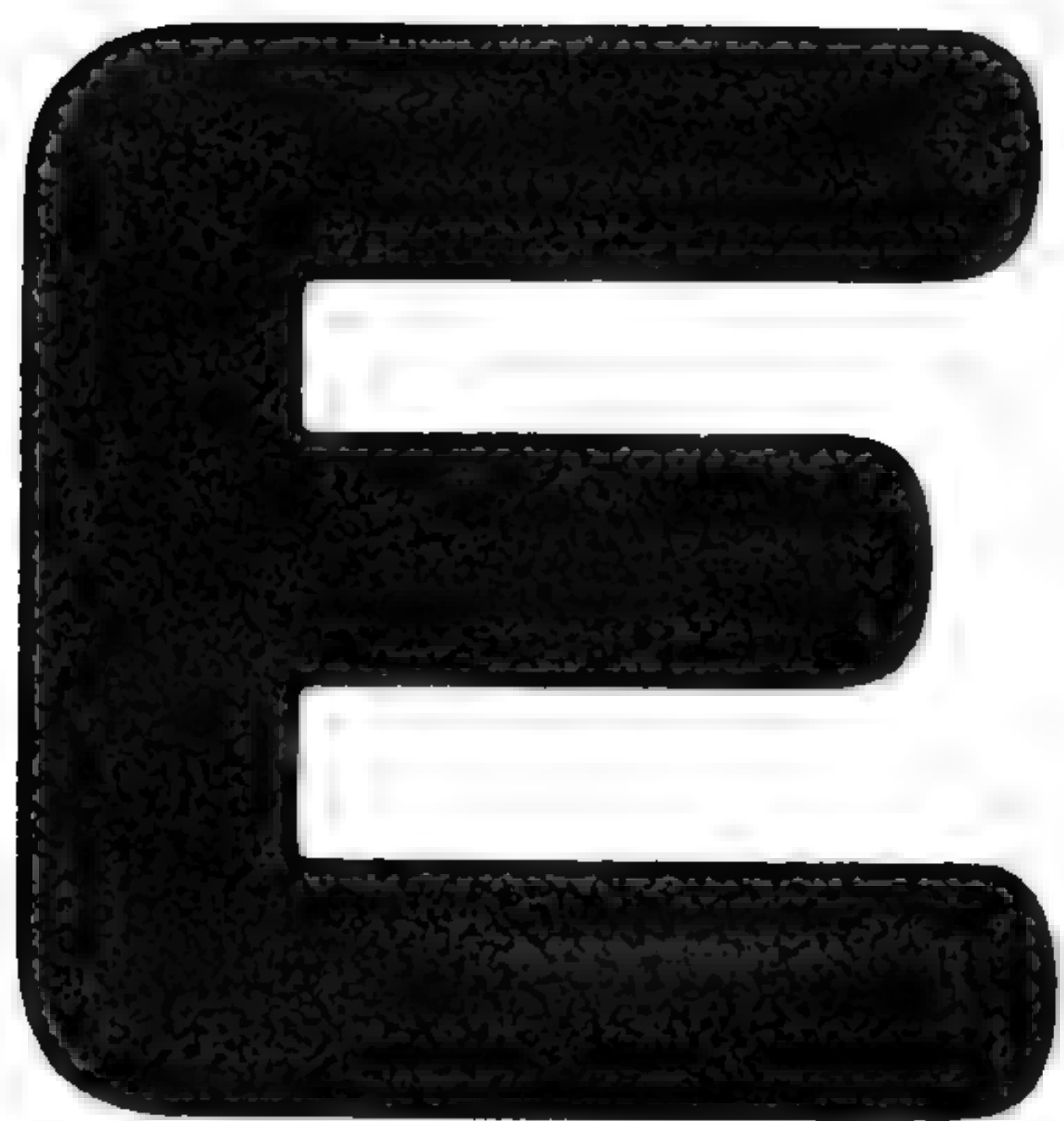
ニーモニックコードをマシン語に変換するアセンブラは、CP/Mだけにあるわけではありません。簡単なものから複雑なものまで、各機種向けにいろいろあります。

実際にアセンブラを使ってみると、どんなことになるのでしょうか。ただ、言葉の意味を知っているという段階から一步を踏み込んで、CP/Mやアセンブラなどを研究してみよう、というのがこのD章です。



# RS-232Cでデータ転送

1	RS-232C活用のハード	194
2	アスキーコードがカギ	196
3	BASICでアスキーコードを操作	198
4	まずはハードのつなぎ方	203
5	RS-232C活用のソフト	205
6	データの転送	208



この頃は、たいていのパソコンにRS-232Cポートとやらいものがついていました。あるいはつけられるようになっていきます。

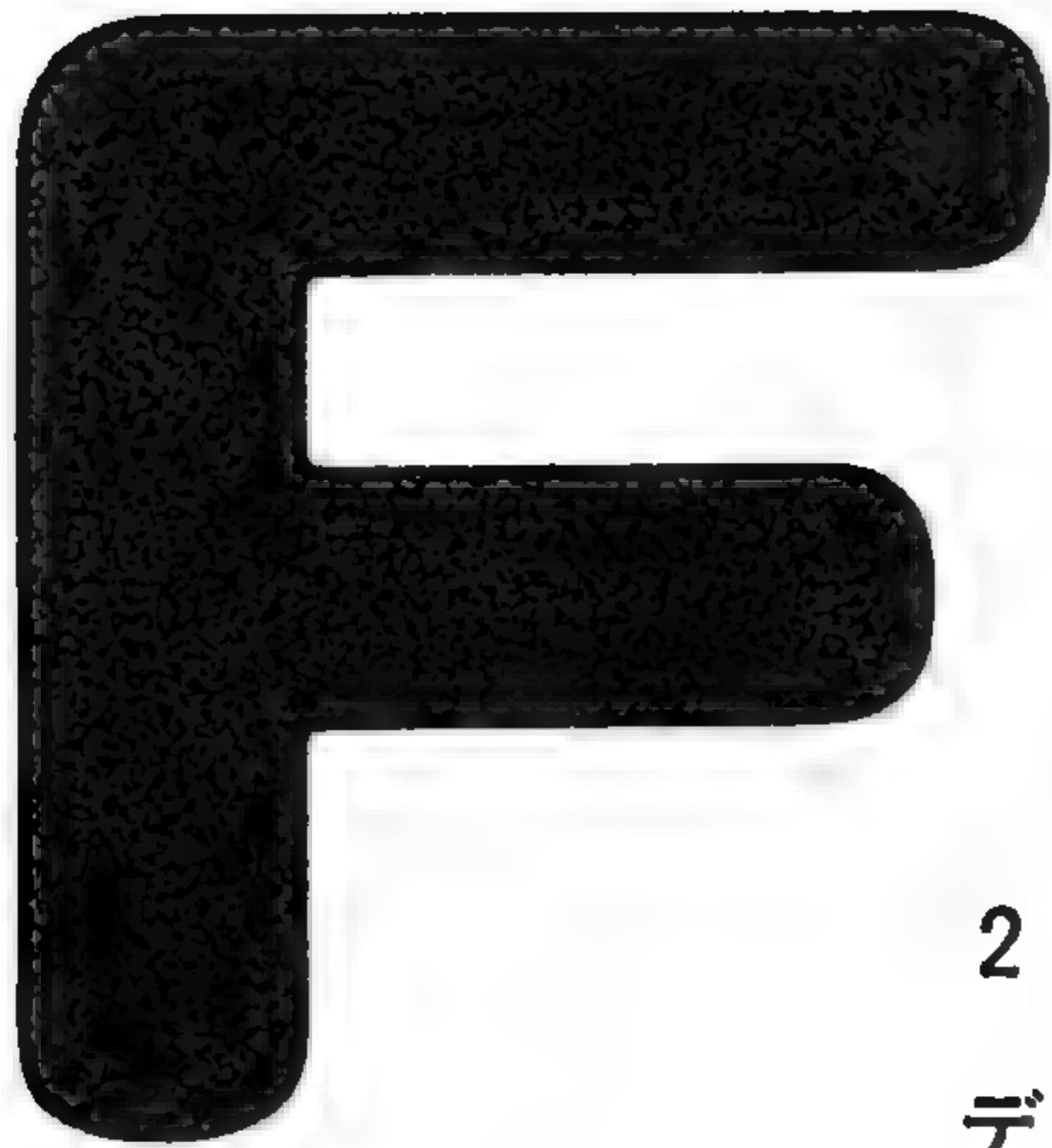
ところが、これがいったいどんな役に立ってくれるのかというと、あまりよくわからないようです。

たとえば、二台のパソコンをつないで、データのやりとりができるとしたら、あなたはどんな活用法、使い方を考えるでしょうか。

こういうことができるようになると、マイコン活用も立派なものになってきそうですね。

ここでは、そのRS-232Cを使つてのデータ転送の基本テクニックを紹介していますので、ぜひ実地への応用を工夫してみてください。これはBASICレベルですが、RS-232CのコントロールのためのBASICは、機種により少しずつ異なりますので、注意する必要があります。





## データ源から直接入力させる法

- 1 工場のハカリとマイコンをつないでみたら 218
- 2 データを取り込むためのプログラム 226
- 3 1／0回路のノイズ対策 231

二台のパソコンを結んで使うという使い方を、さらに徹底して追究すると、どういうことになるでしょうか。

実際にパソコンを活用しようとして、ソフトをつくり、実地に使うとします。いったい何が問題になると思いますか。結局は人間がやらなければならないデータ入力でしょう。

そこで、もうちょっとがんばって、データ入力も機械からパソコンへ直接やってしまう方法はないだろうか、と考えました。

この最後のF章では、実際にやっているひとつの例を示しています。これも応用はかなりできるのではないのでしょうか。こうなるとハードの工作とか発注とかいう問題も出てきて、お金もかかりますか、これは8ビットマイコンの活用を極めた、ひとつの姿として参考にしたいのです。

絵とき\*笹倉正義

カバーデザイン\*井上丈児



……ざつと以上のような構成になっています。一応、初心者から、ある程度経験している人まで、それぞれのレベルに応じて、幅広く役立てていただけるのではないかと思います。

具体的なパソコンもいくつか出てきますが、これはメーカーや機種を限定した本ではありません。どんな機種をお持ちの方にも参考になるはずです。ただし、CPUは最も広く普及している80系・280を中心にして説明しています。

本書の中で、みなさんをご案内する友子ちゃん（さん、というべきでしょうか）は、とあるメーカーに勤務しているOLです。

彼女は大変な勉強家で、ここ数年にわたり、コツコツとマイコン研究に打ち込んできました。会社の制服らしきものを着て登場することが多いのは、彼女がその研究の成果を認められて、社内の小さな研修会などのインストラクターをやったりしていたからでしょうか。

どうぞ、みなさんも彼女の話聞いてやってください。

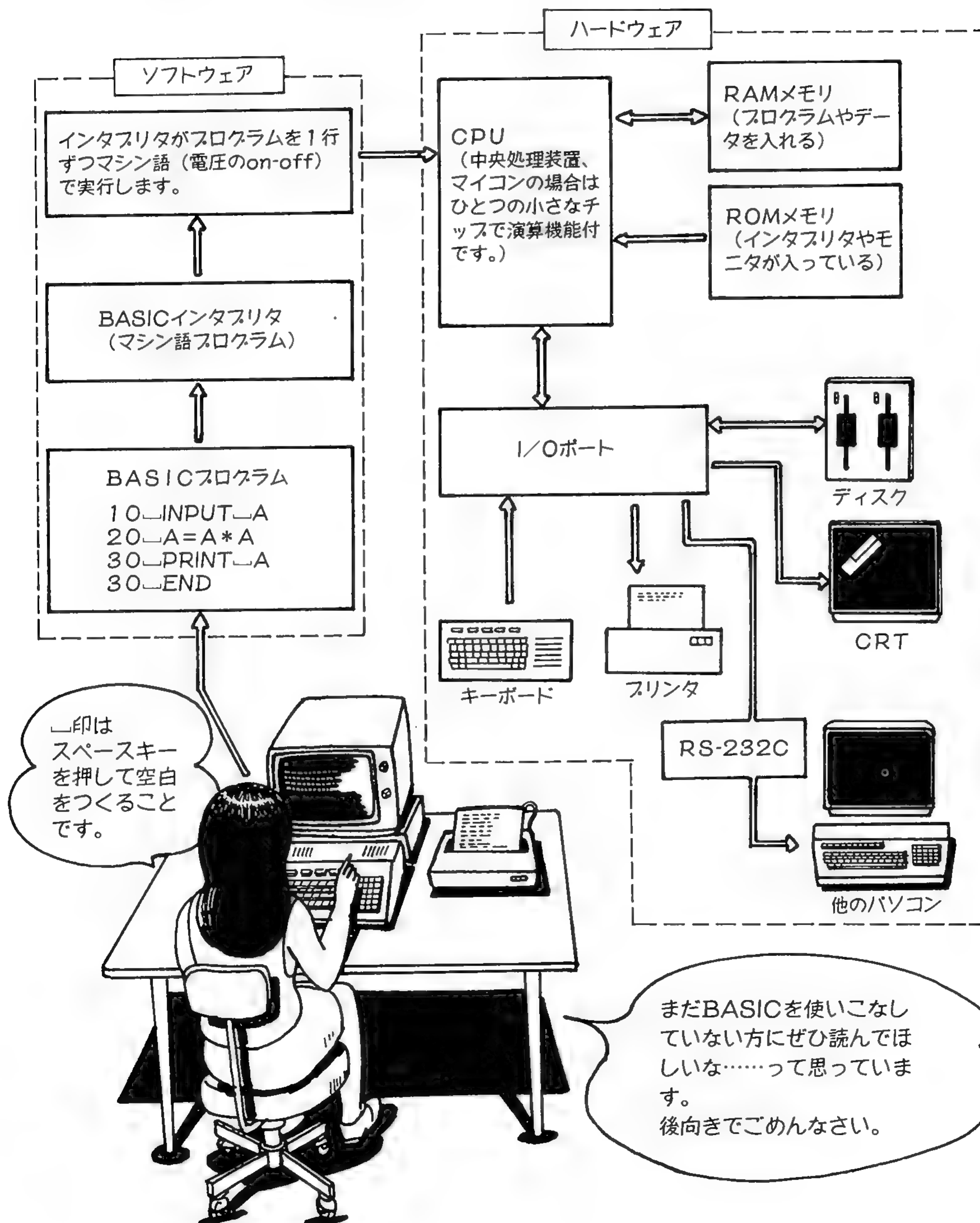
昭和59年8月

著者



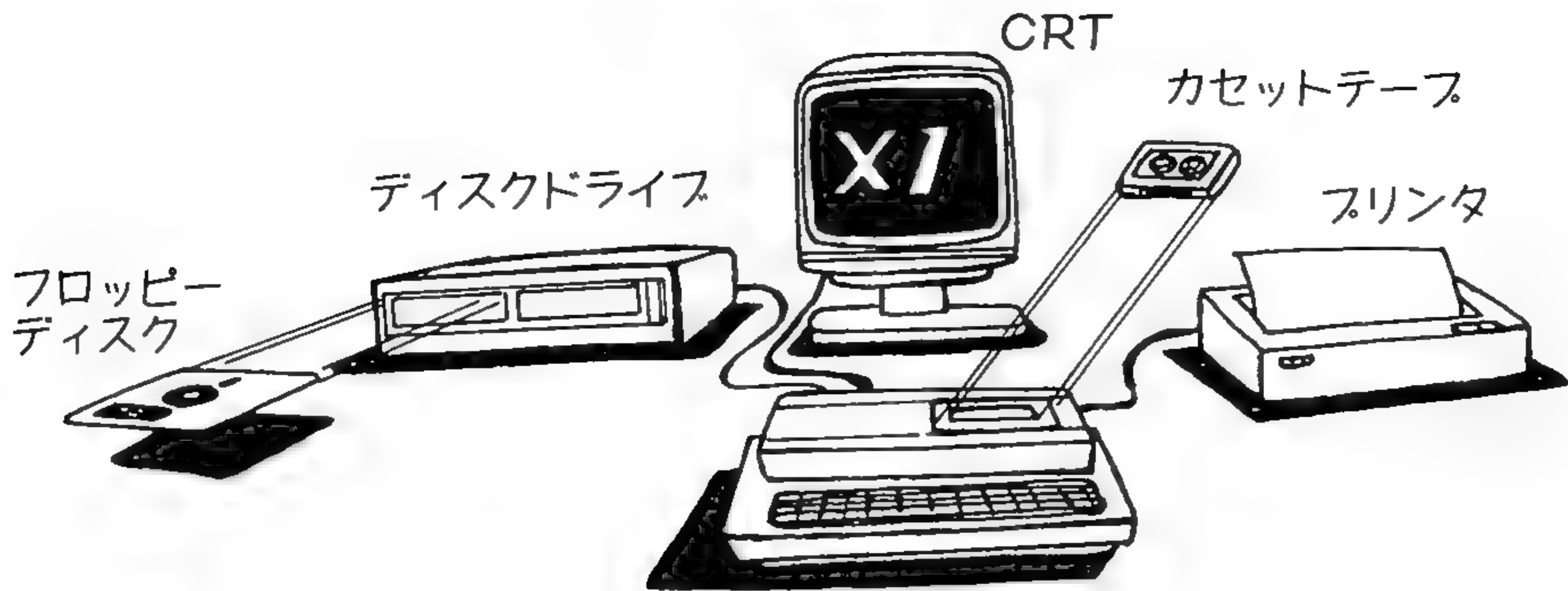
# A BASICはパソコン理解の第一歩

‘パソコン’ というと ‘BASIC’ となるのですが、実際には、それぞれのCPUのマシン語で高速に実行しているのです。ここではまずBASICのおさらいをしておいてB章以降でマシン語の研究をしてみたいと思います。





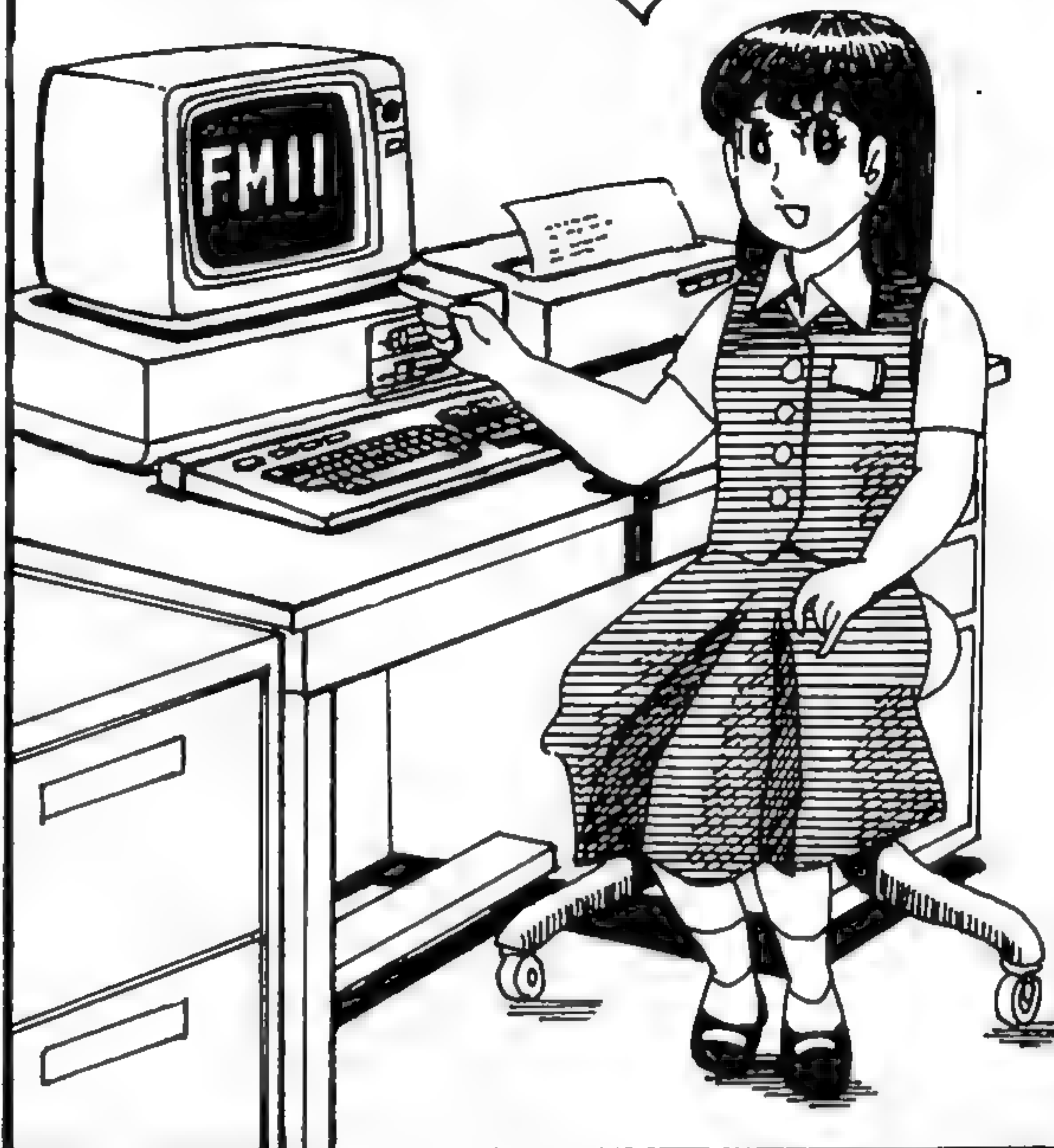
# ①パソコンはBASICが使える





ワンボードマイコンは一部のマニアの間でしか普及しなかったのに、パソコンがここまで広がってきたのは、価格もさることながら、その使いやすさの違いです。ワンボードマイコンは機械語しか使えませんでした、パソコンでは人間の言葉に近いBASICなどの高級言語が使えるからです。

何年か前までは個人的に持てると言ったらワンボードマイコンだったのですが今では価格も安くなってパソコンが持てるようになりました。



ワンボードマイコンとパソコンの大きな違いは、そのメインメモリの容量の大小と言えます。

64Kバイトメモリ

RAM 256バイト

ROM 2Kバイトメモリ

ワンボードマイコン

パソコンでは、10進数、アルファベット、カタカナ、漢字などが使えます。そしてBASICなどの高級言語も使えるというわけです。

たとえばワンボードマイコンでは扱えるのは16進数だけで0~9、A~Fしか使えませんでした。

パソコンではメモリがたっぷり使えるので、ワンボードマイコンではできなかったことができるのです。

0000

モニタプログラム	OSプログラム	高級言語翻訳用プログラム	高級言語プログラム	機械語プログラム	データ、変数、スタック
----------	---------	--------------	-----------	----------	-------------

たとえばメモリをこのように使うことができます。(メモリマップ)

うーん、ぎっしり詰まってる。

Xメモリ

FFFF

FORTRANはIBMから発表された世界初の高級言語処理プログラムです。

コンパイラを使う代表的な言語はFORTRAN (フォートラン) です。

高級言語はそれだけでは機械にわかりませんから、機械にわかる言葉に翻訳するプログラムが必要で、その形式によってコンパイラとインタプリタに分けられます。

コンパイラもインタプリタもその役目はプログラムの翻訳・通訳です。

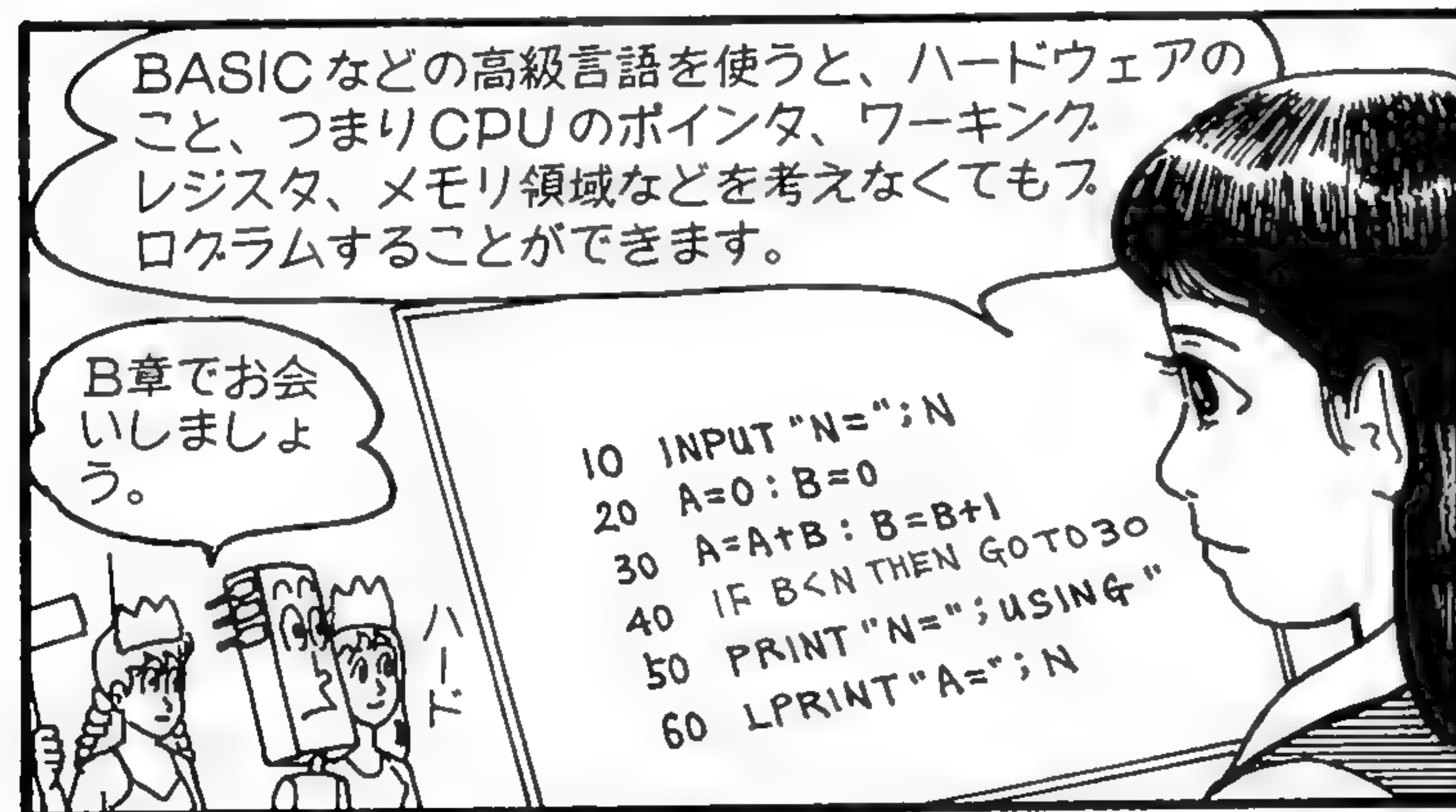
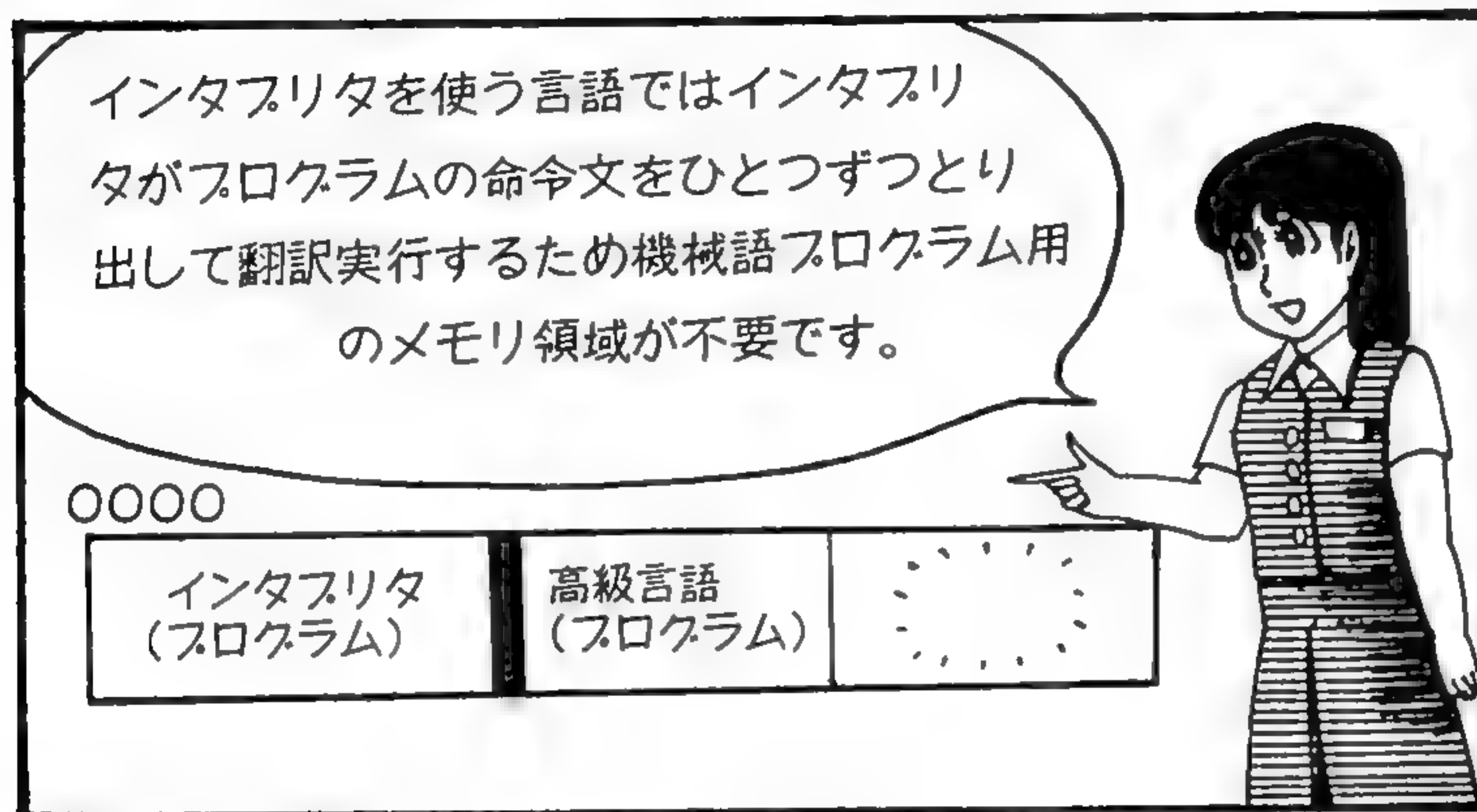
FORTRAN プログラム言語の例

```
* EX TASHISA
  INTEGER
  REAL H;
  S=0.0
  READ(5,*)
```

FORTRAN 77

これがソフトウェアちゅうもんらしい。





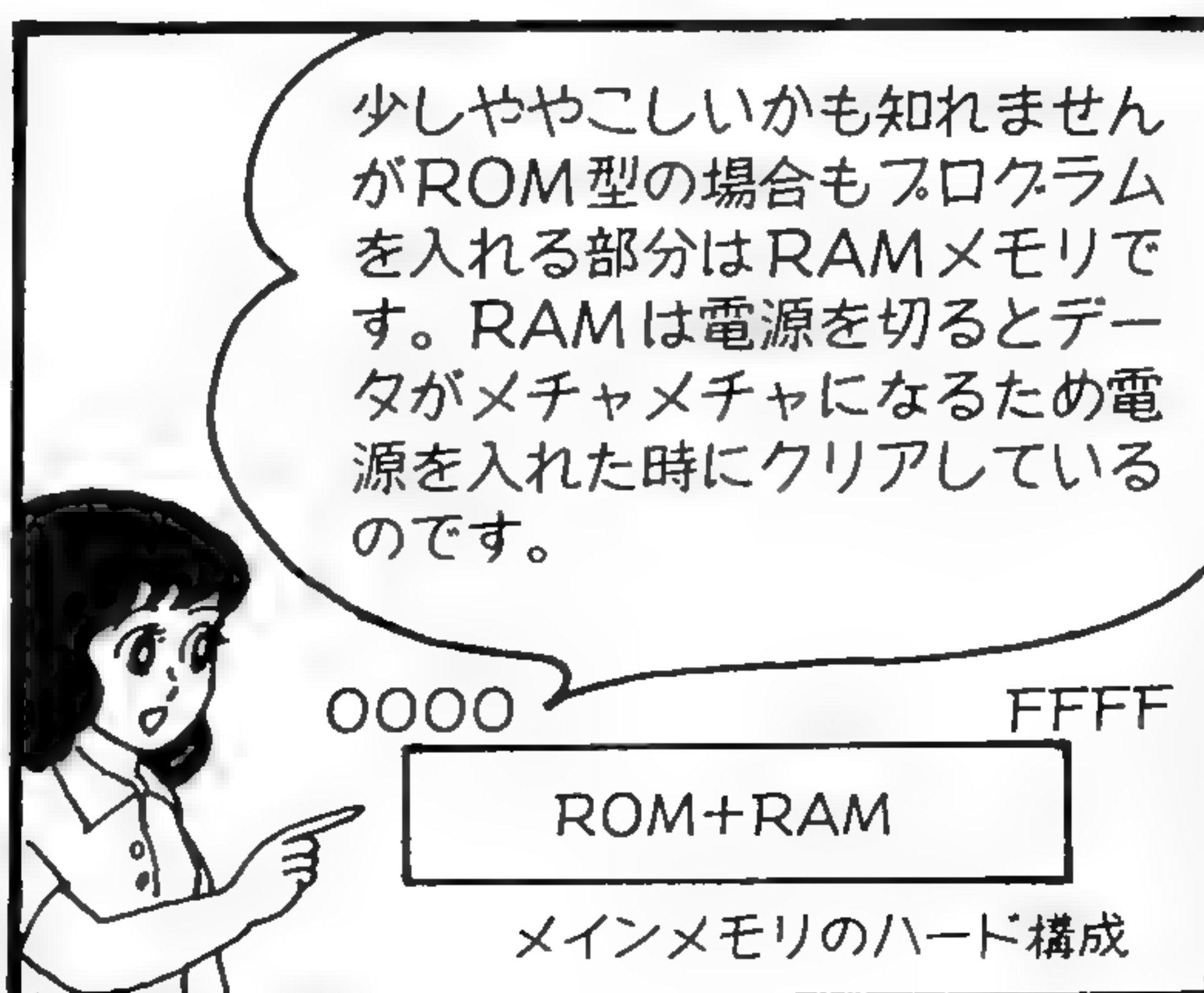
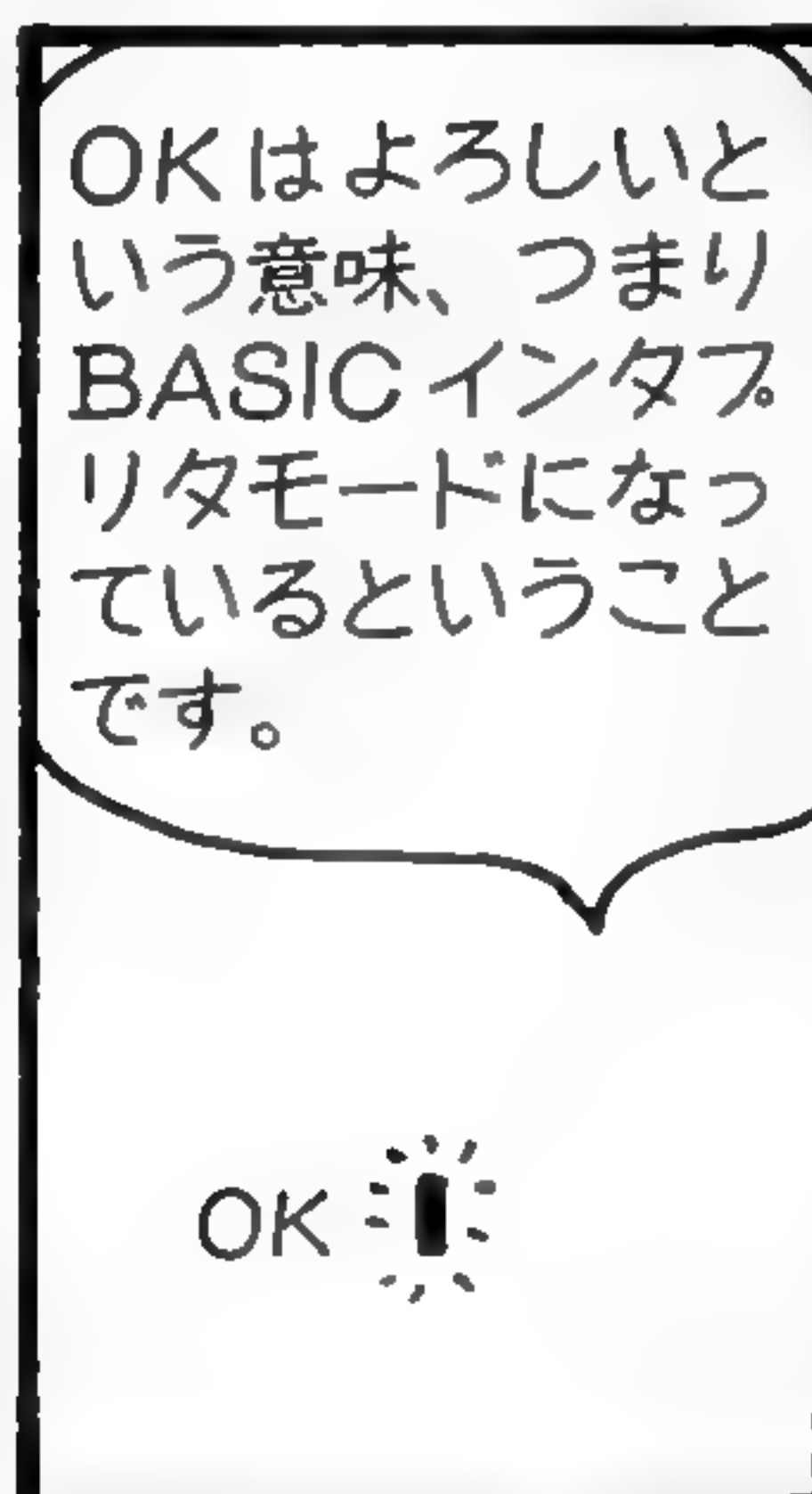
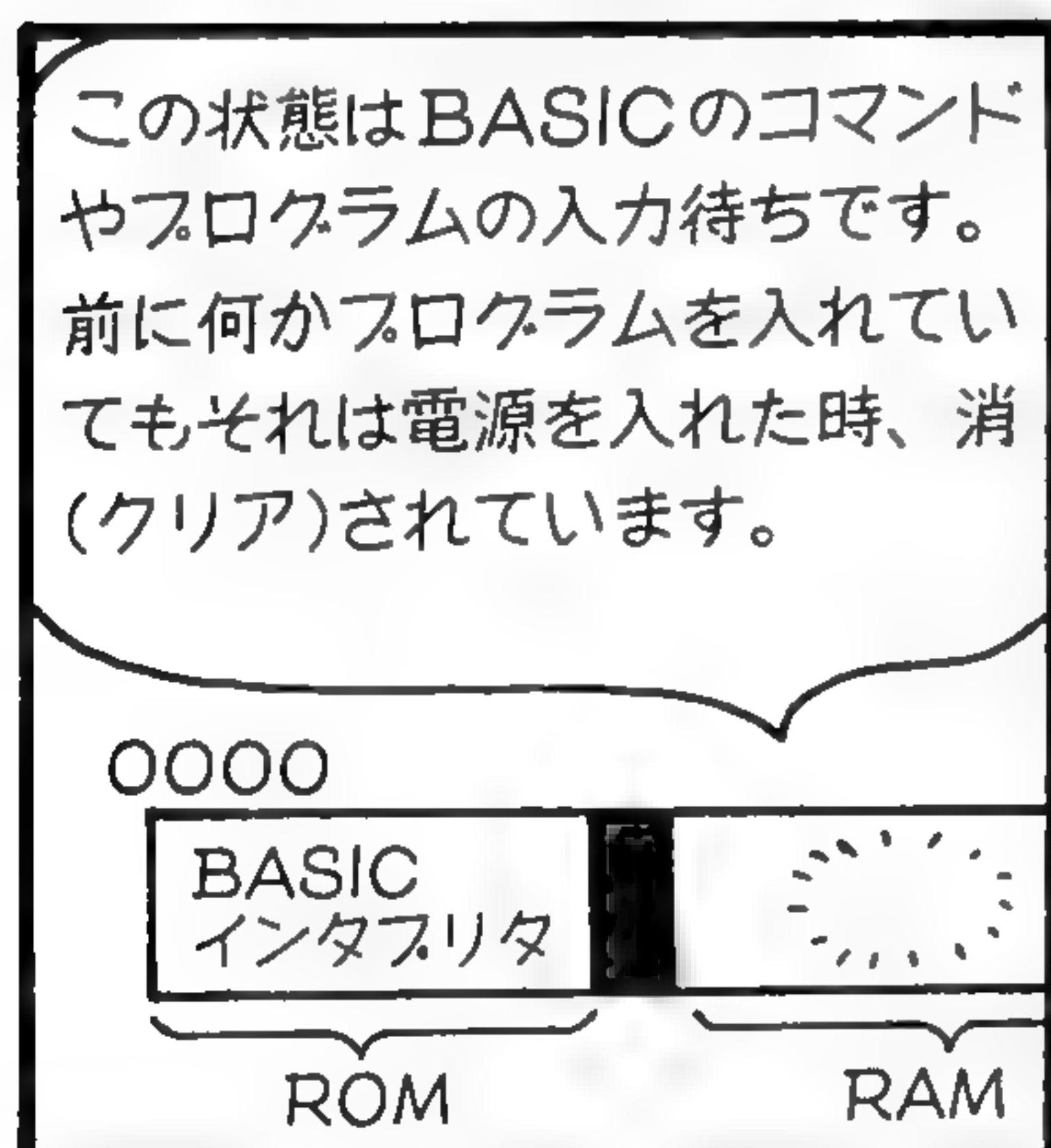
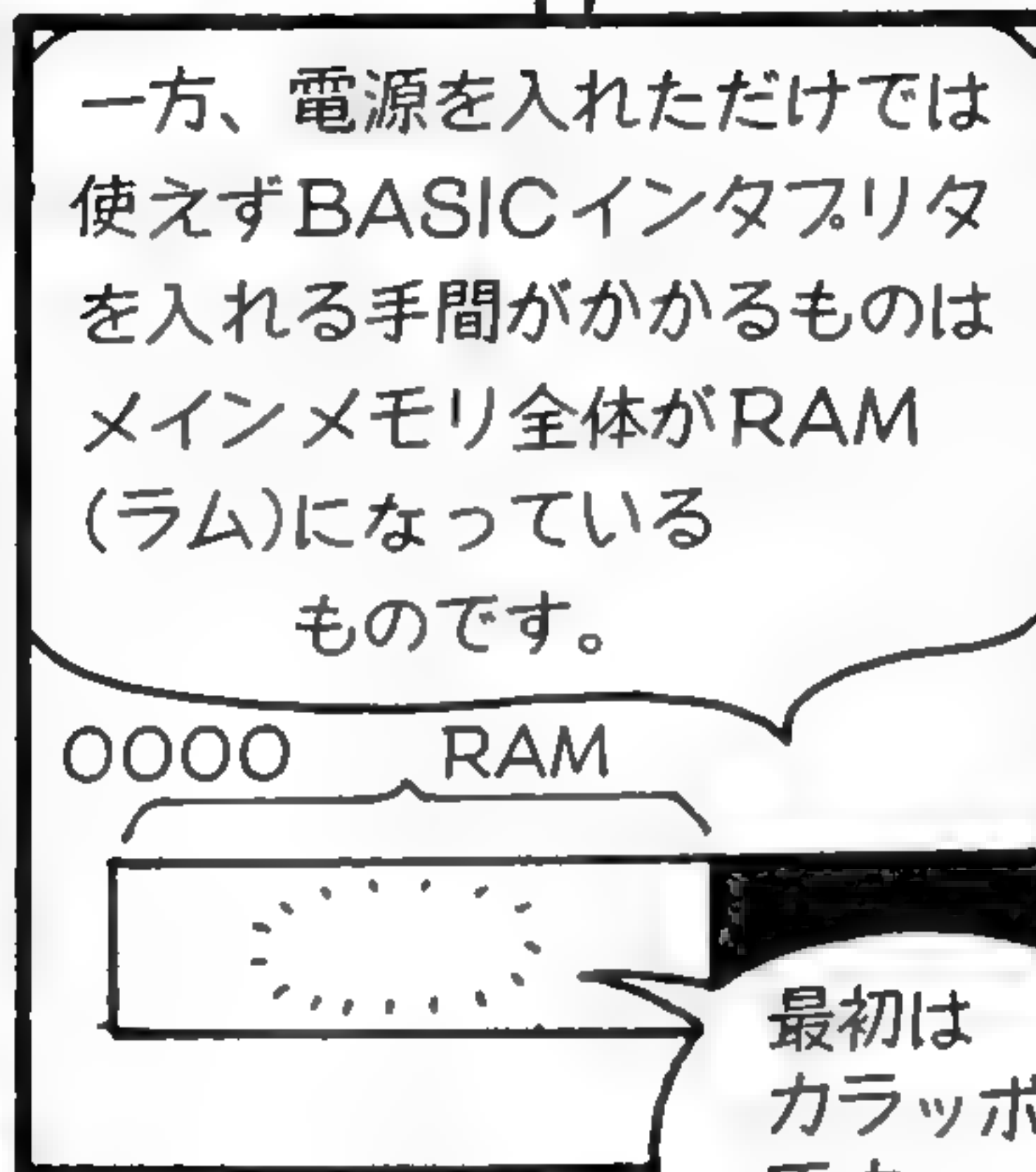
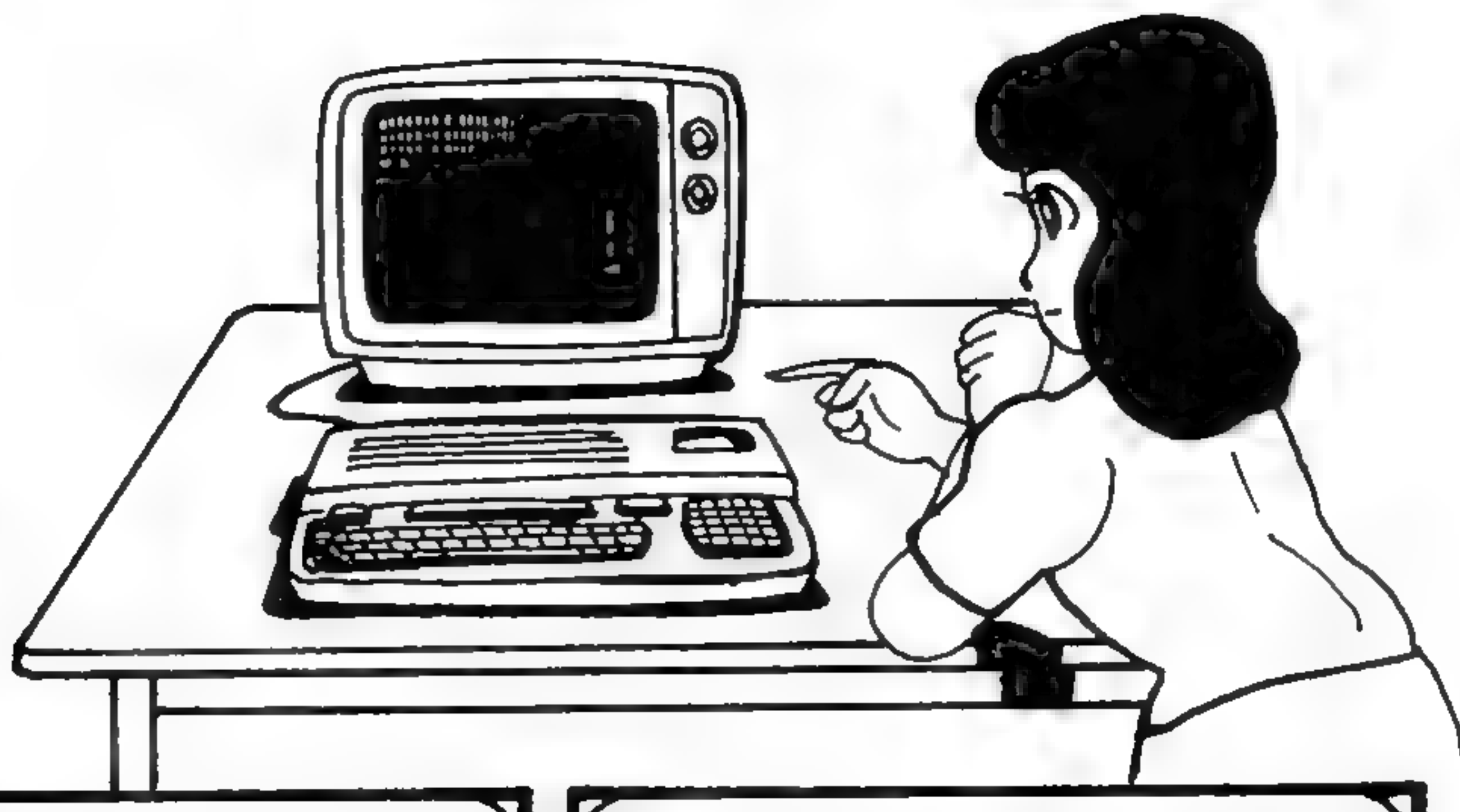
たとえばBASICではコマンド、ステートメント、ファンクショ  
ンの3種類に分類されるたくさんの記号（BASIC言語）がありま  
す。BASICでパソコンを動かそうと思ったら、このBASIC言語を  
覚える必要があるわけです。

コマンド(命令)	ステートメント(文)	ファンクション(関数)
AUTO	COLOR	ABS
CLOAD	DATA	ATN
CSAVE	DIM	COS
LIST	END	INT
MON	FOR	LEN
RETURN	NEXT	SIN
RUN	GOTO	USR
⋮	⋮	⋮





## ②ROM型パソコンと オールRAM型パソコン





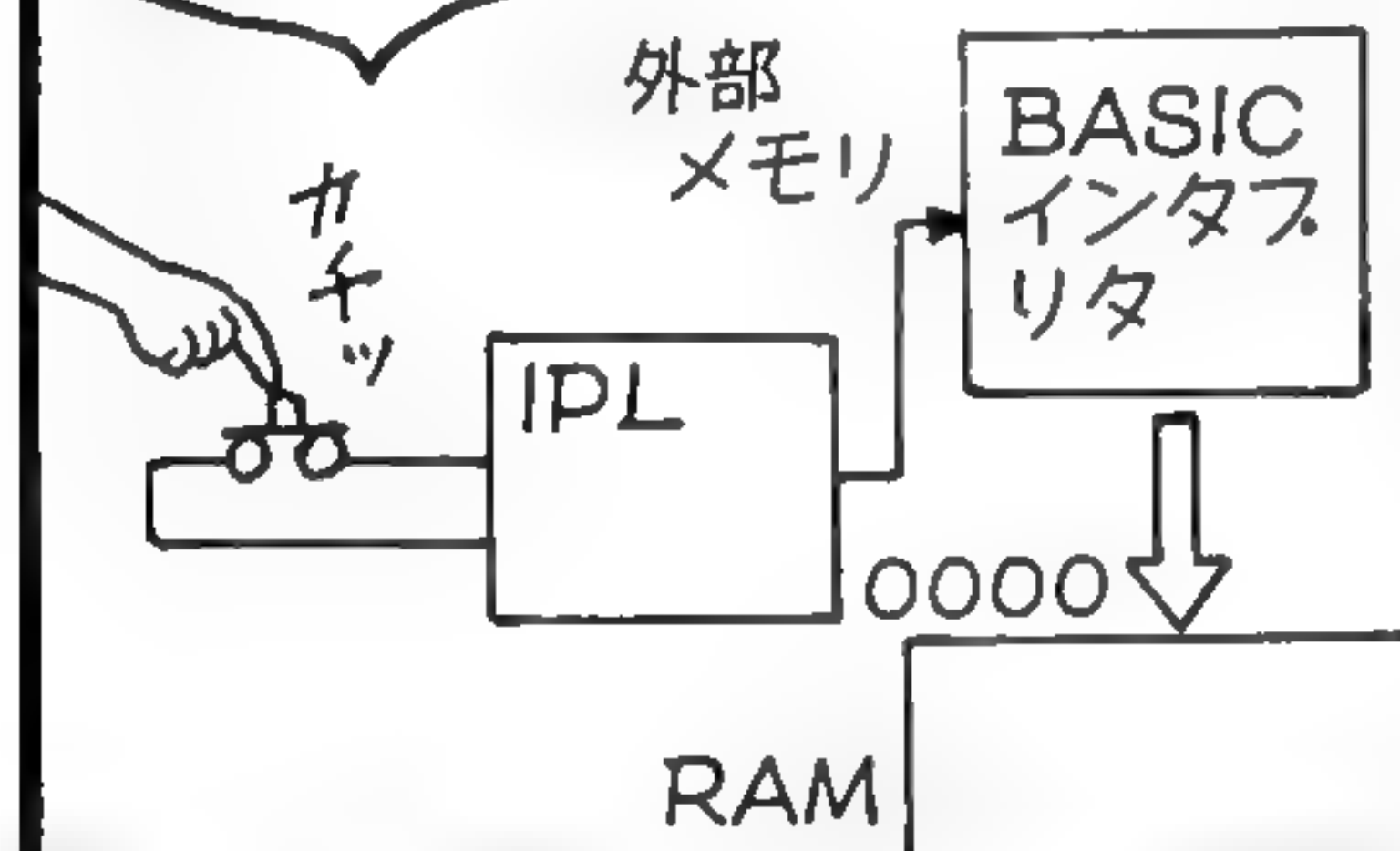
そこでインタプリタを入力するための外部記憶装置が必要です。カセットテープやフロッピーディスクが使われます。



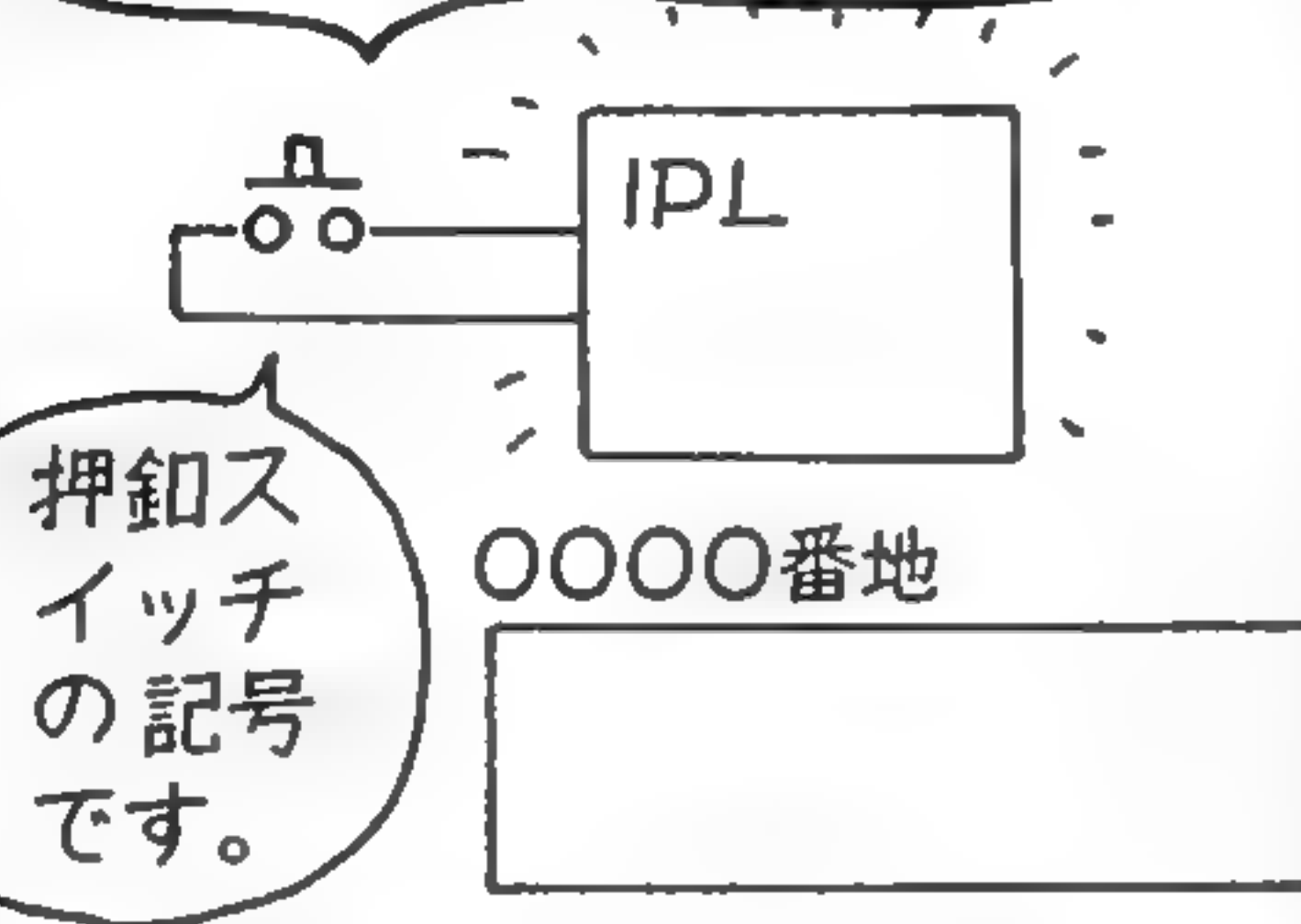
次はオールRAM型パソコンです。これは電源を入れた時すべてのメモリエリアがクリアされそのままではどのプログラムも走りません。



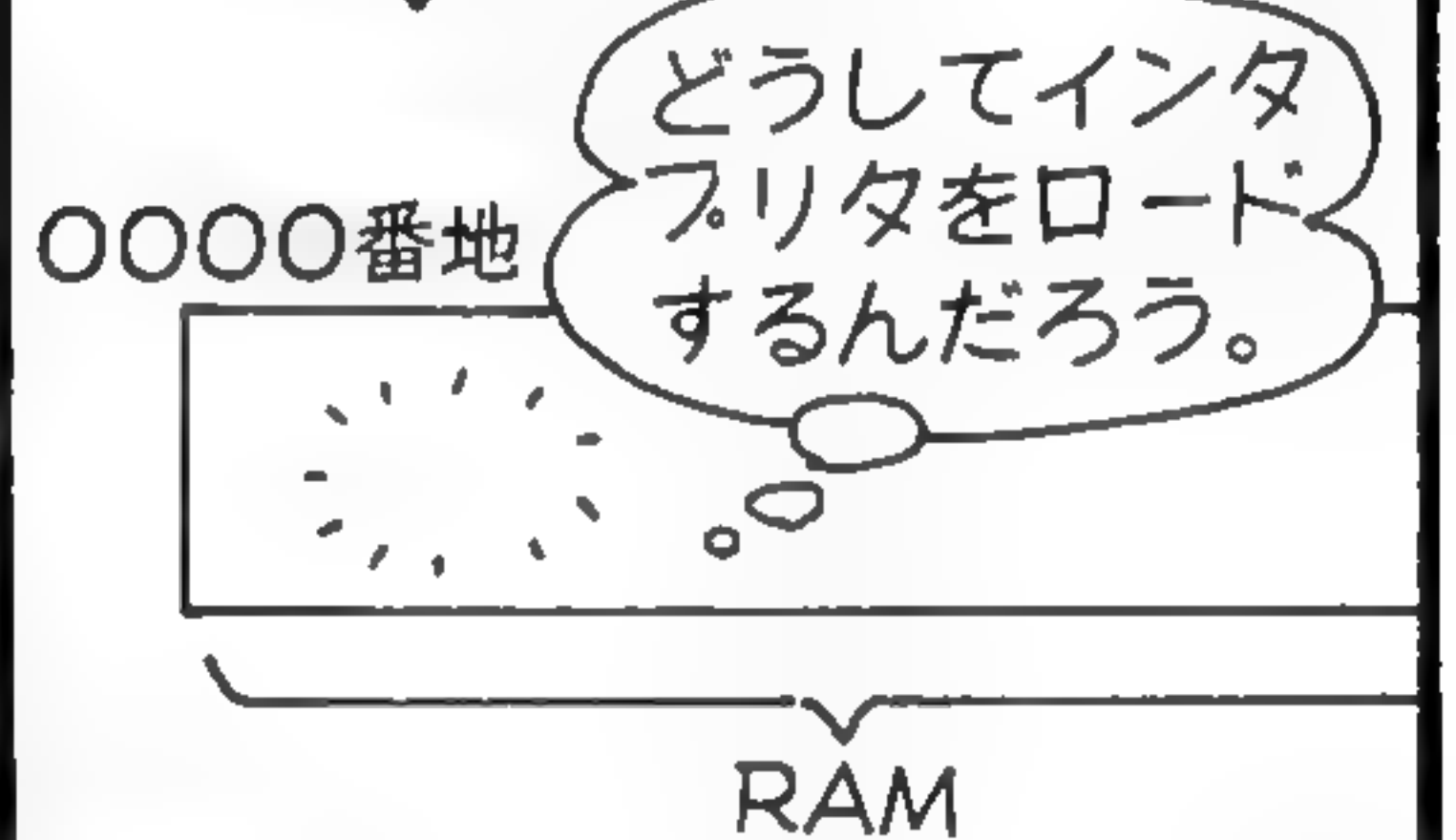
このIPL押釦を押すと、外部記憶装置からBASICインタプリタがメモリにロードされます。



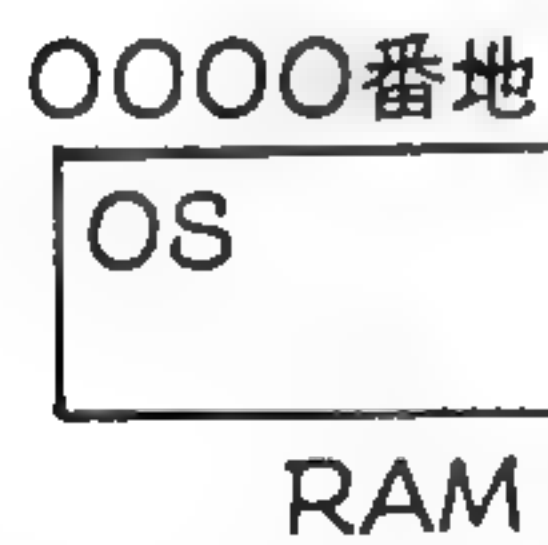
そのためIPL（イニシャル・プログラム・ローダ）という機能が必要になります。



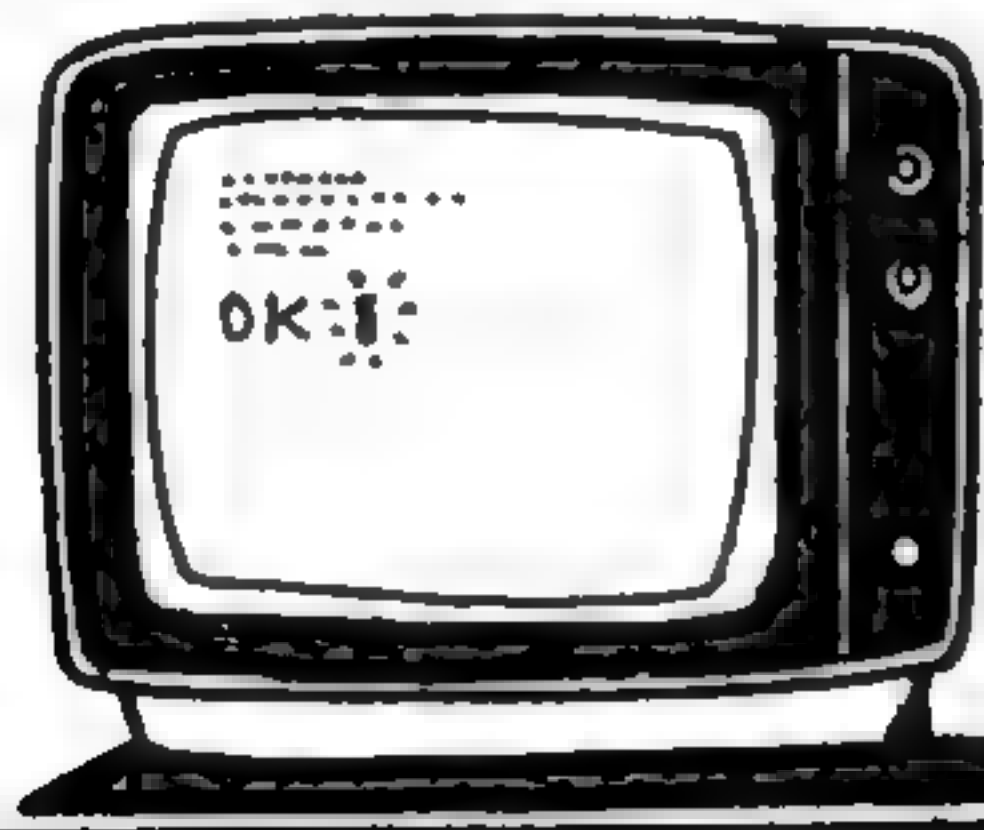
オールRAMにするためには0000~FFFF番地の中には何もプログラムしておけません。



それはオールRAMの場合、ここにオペレーティング・システム・プログラム（OSといいます。）を入れることができるからです。



これでROM型パソコンと同じようにBASICが使えるのですが、なんでこんなめんどろなことをするのでしょうか。



ロードが完了したらIPLは見かけ上姿を消して、BASICモードになります。



このCP/Mを使うとBASICの他の言語のプログラムも使える利点があるのです。それはOSというのは言語開発にも使えるからです。話がむずかしくなりました。このCP/Mの使い方の例はあとの章ででてきます。

CP/Mはデジタル・リサーチ社の登録商標です。



たとえば8ビットCPUのひとつである8080のOSとしてはデジタル・リサーチ社のCP/M（シーピーエム）があります。

CP/Mは8ビットCPUのOSの主流になりつつある。





CP/Mマシン（CP/Mの走るパソコン）ではIPLスイッチを入れると高速でフロッピーディスクのCP/Mプログラムがメモリにロードされます。



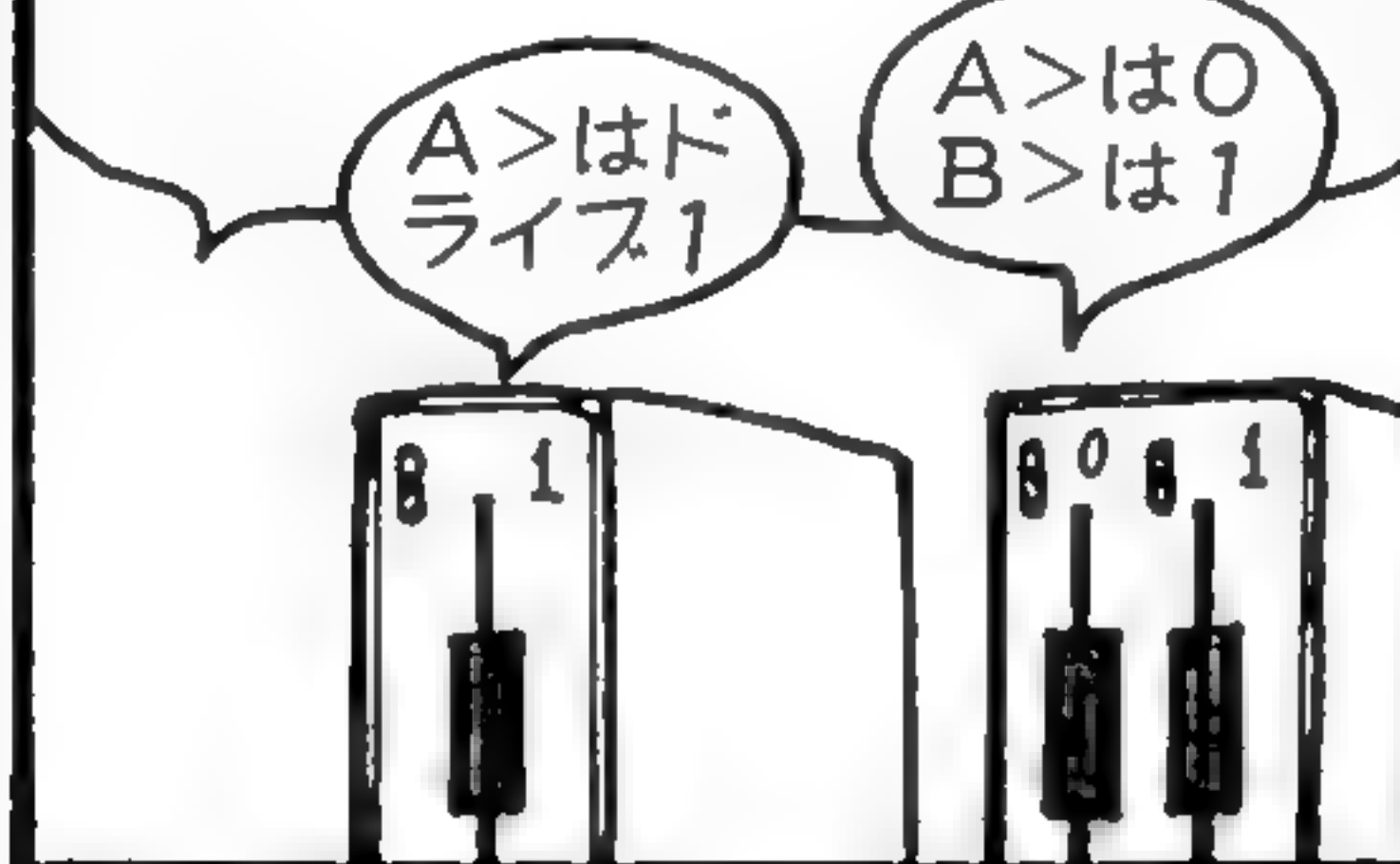
CP/Mが走る（なぜかこういう言い方をします。）パソコンは少なくとも1台のディスクドライブが必要です。CP/Mではカセットテープは使わないのです。



CP/Mのモードになったら、こんどはキーボードからBASIC [RETURN] と入れてやるとフロッピーからBASICインタプリタがメモリにロードされます。

CP/M      BASIC  
インタプリタ

A>のAはドライブのNo.を示しています。日本のパソコンのドライブは0、1か1、2というNo.ですけどね。



BASICモードは\*印やOKですがCP/MではA>というように表示になります。

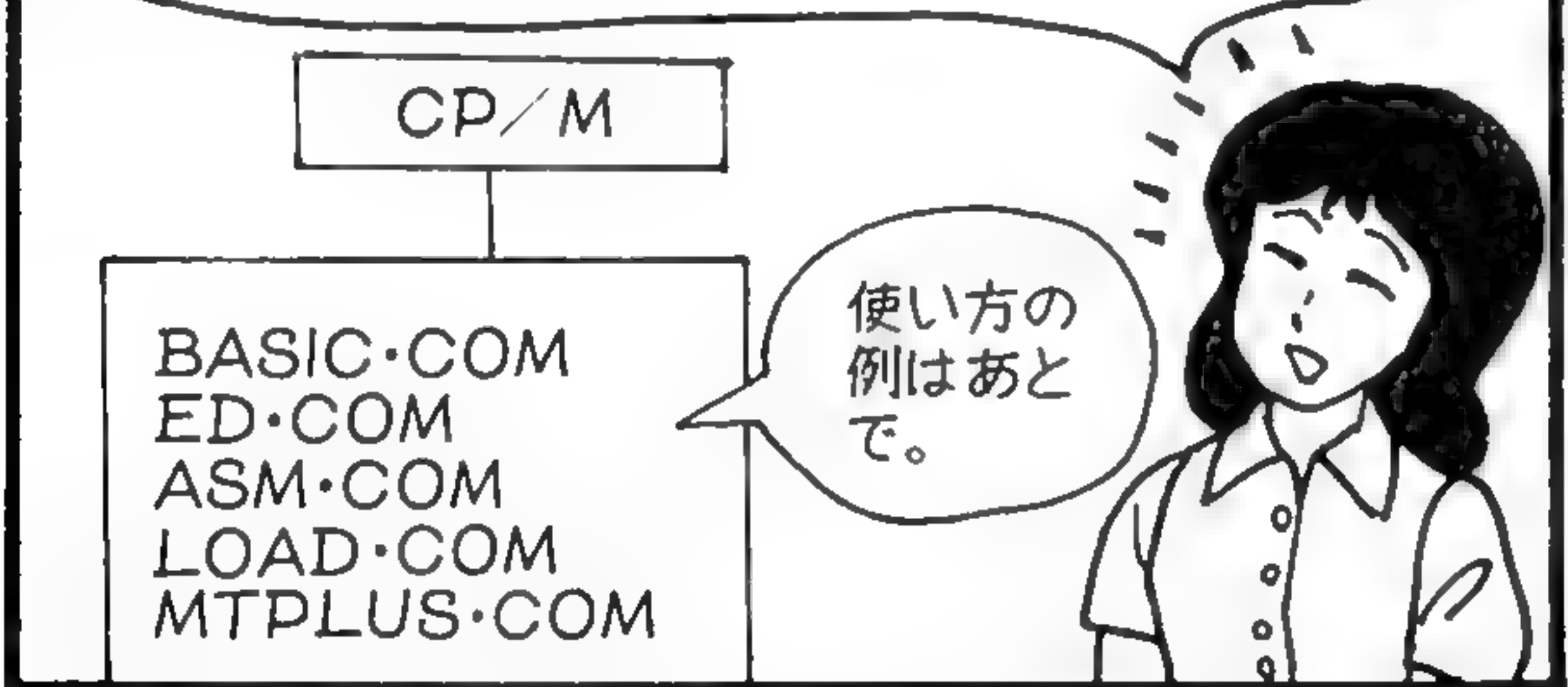


今はBASICインタプリタを入れましたがこの他にもいろんなプログラムが使えるようになっています。

CP/M

BASIC.COM  
ED.COM  
ASM.COM  
LOAD.COM  
MTPLUS.COM

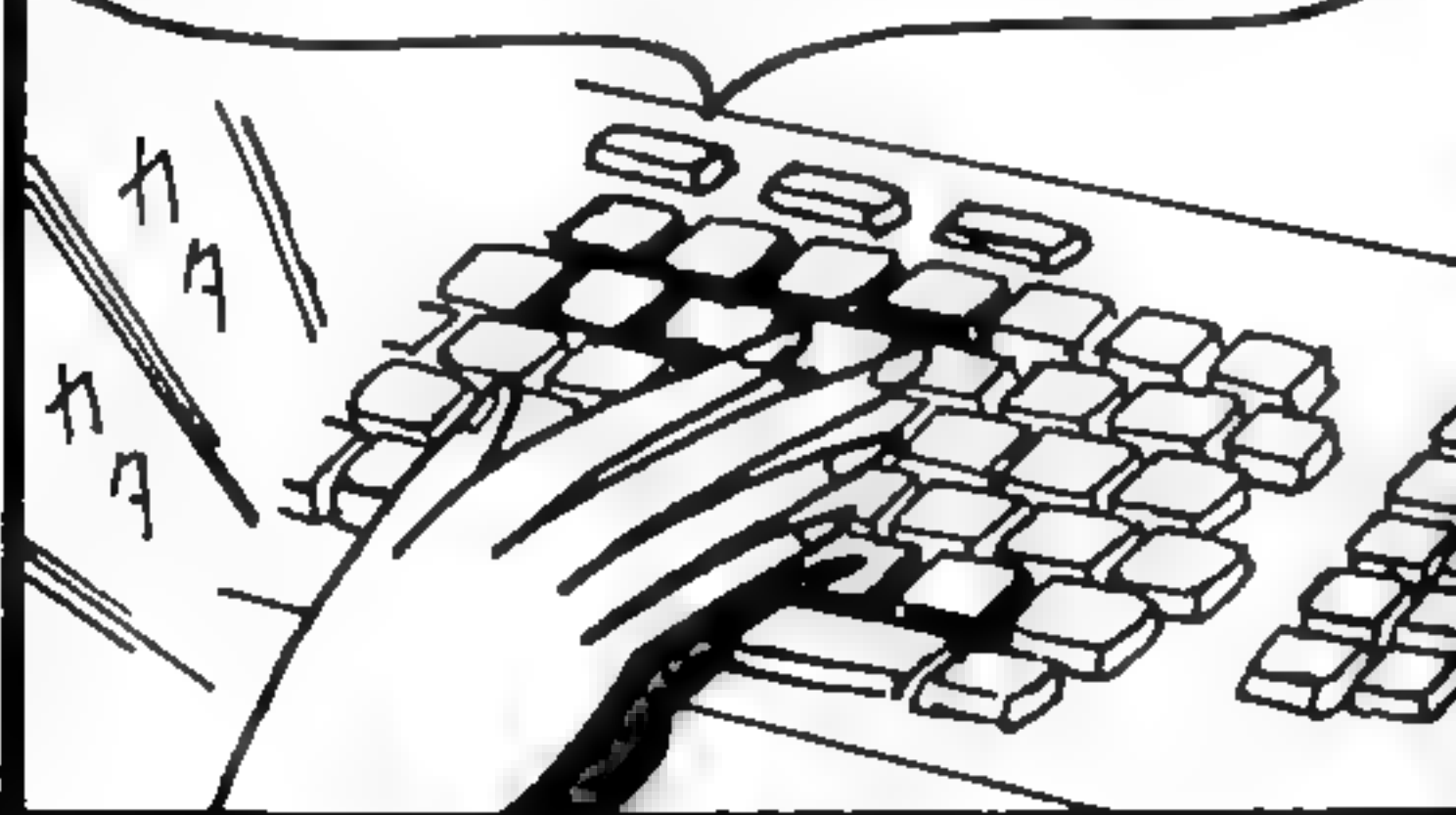
使い方の  
例はあと  
で。



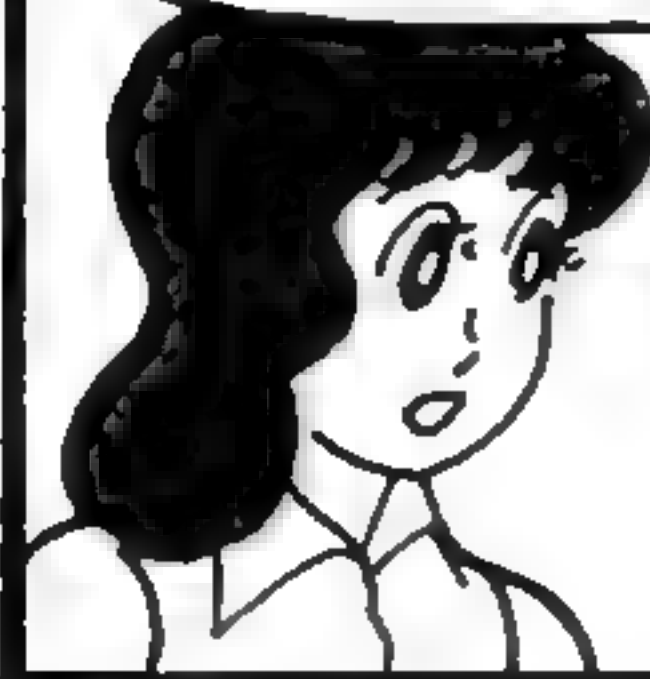
ロードが完了するとBASICモードになりCP/Mは見かけ上消えてしまいます。



では、BASICでプログラムしてそれを走らせてみましょう。マニュアルを見てるだけではどう使っているのかわからない記号がいっぱいありますから。



BASIC言語だけで満足ならばROM型のパソコンが便利だし、成長に合わせて別の言語を使いたいのだったらオールRAM型パソコンを選ぶべきでしょう。



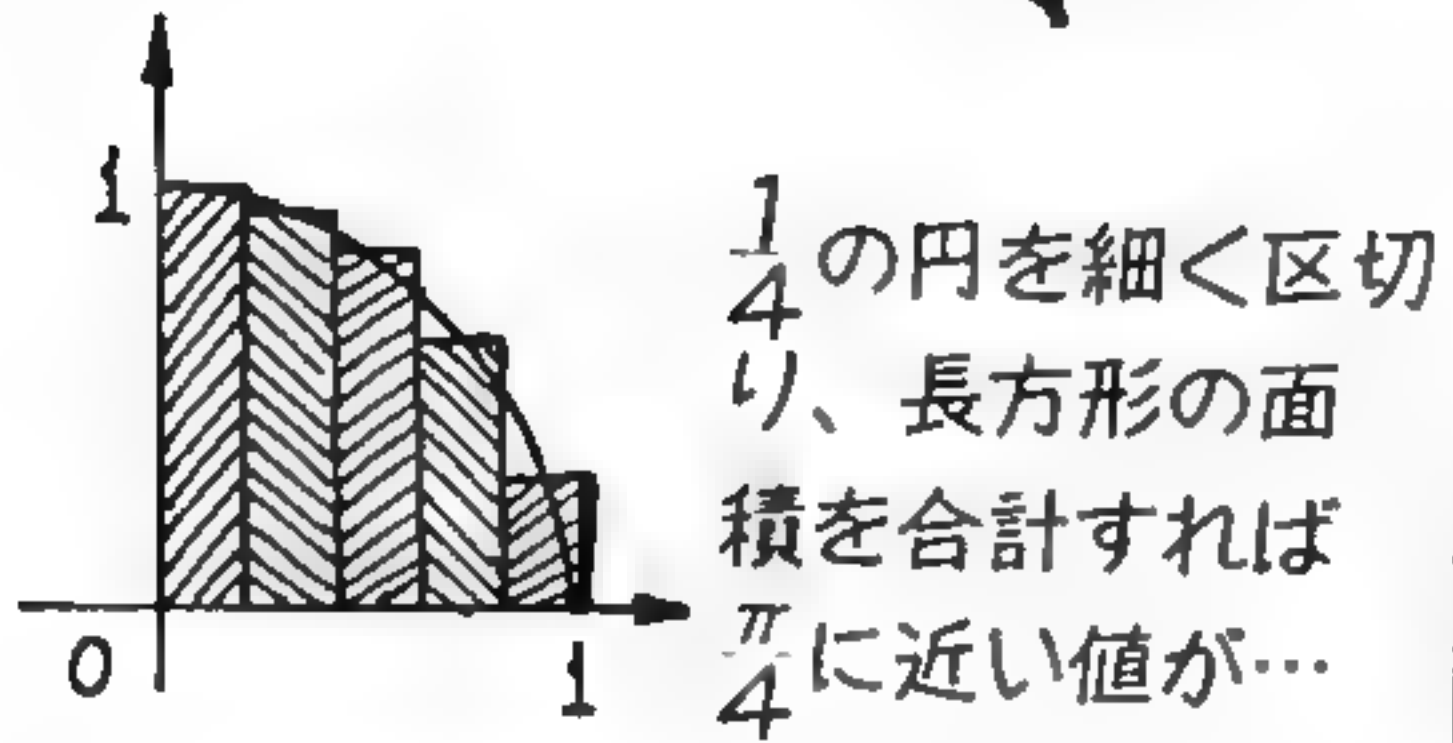
BASIC  
だけかどうか  
それが問題

つまり、オールRAM型ではBASIC以外の言語やソフト開発（マシン語レベルのプログラムづくり）ができるということです。





次に繰返しルーチンのFOR～NEXT文の使い方の応用として半径1の円の面積から $\pi$ (3.14)を求めるプログラム。

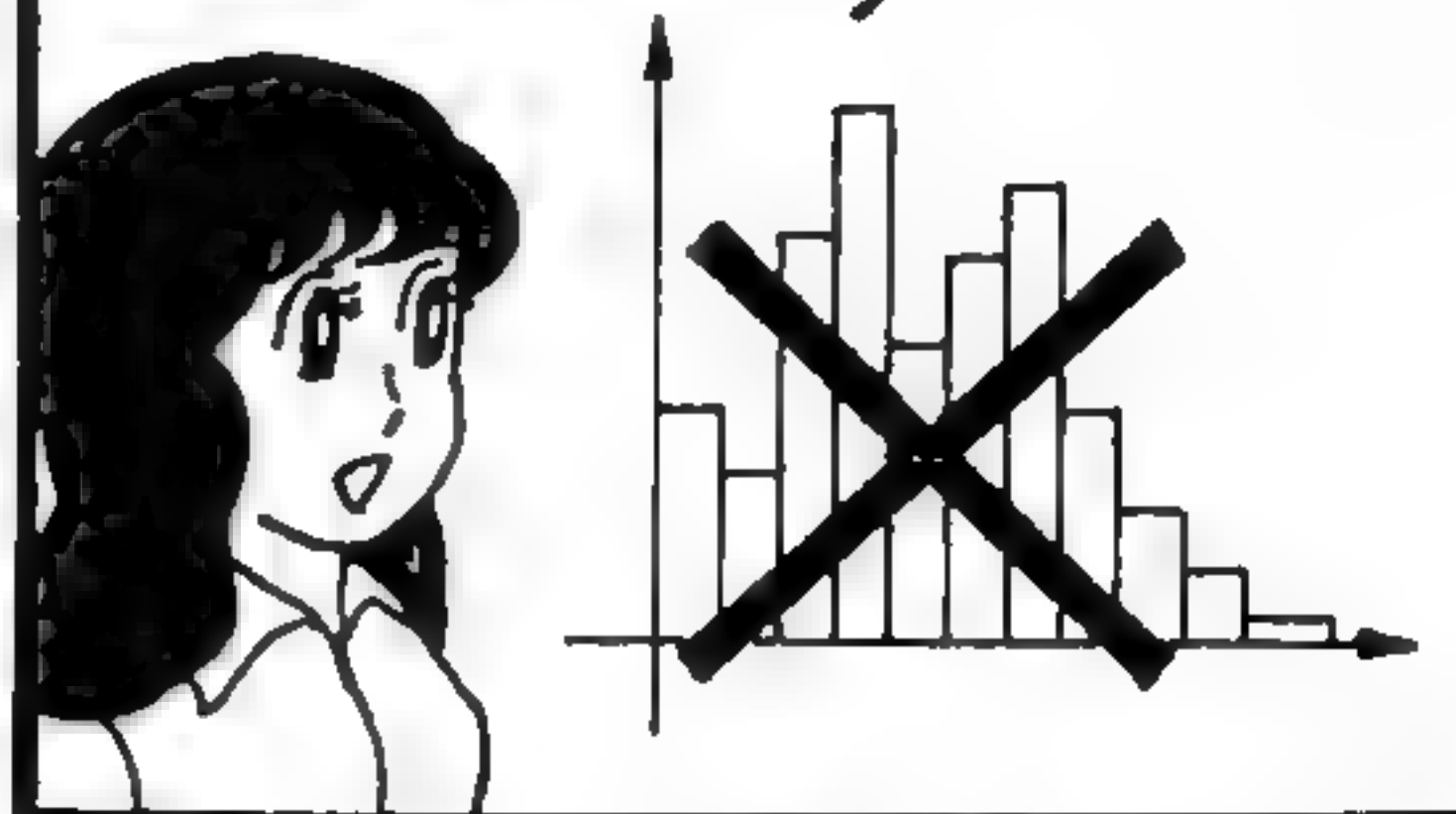


用意したのはまず1からNまで1ずつ増加してそれを累計していくプログラム。

$$A = 1 + 2 + \dots + N$$

1から順に1ずつ増してNまでの累計Aを求めます。

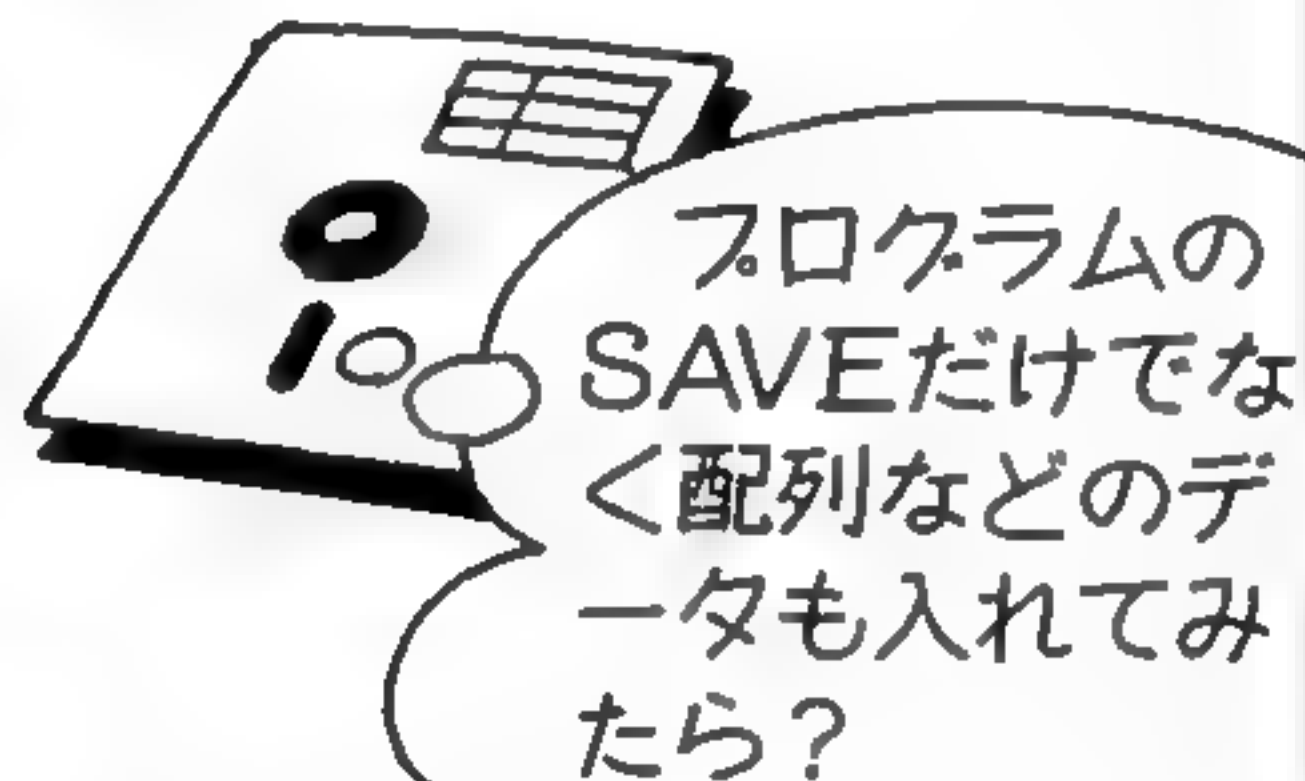
その前にちょっとおことわりしておきますが、ここではグラフィック関係のプログラムは用意していません。



これらの例題でかなりのステートメントやファンクションを使いますのでこれまでチンプンカンプンだったマニュアルが読めるようになることはうけあいです。



最後にフロッピーを使って、データを保存するプログラムを用意しました。



それから、表計算にとっても便利な配列を使ったプログラム。ビジネス向ソフトではなくてはならぬものです。

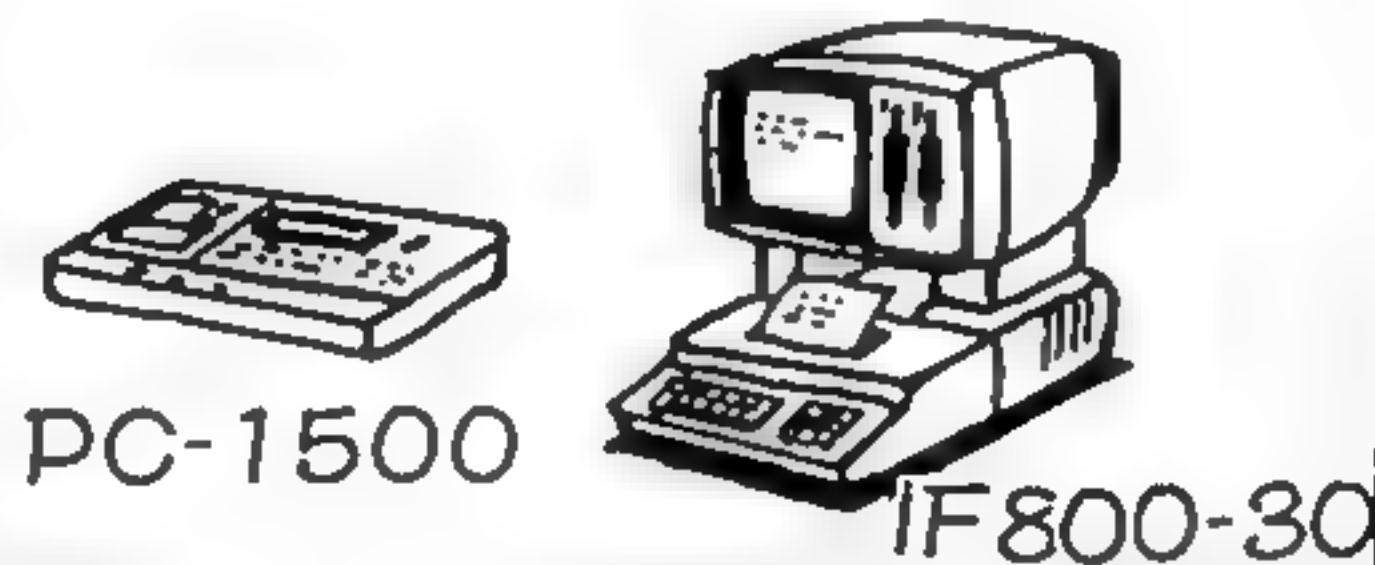
DIM A(3, 20) とかで表現される配列と、実際の表のあやしい関係をスクープします。

また、キーボードはメーカーによって違います。たとえば、データの区切りとして使うRETURNキーにしても、

OK AUTO  
表示 → 10

キーイン → A U T O RETURN

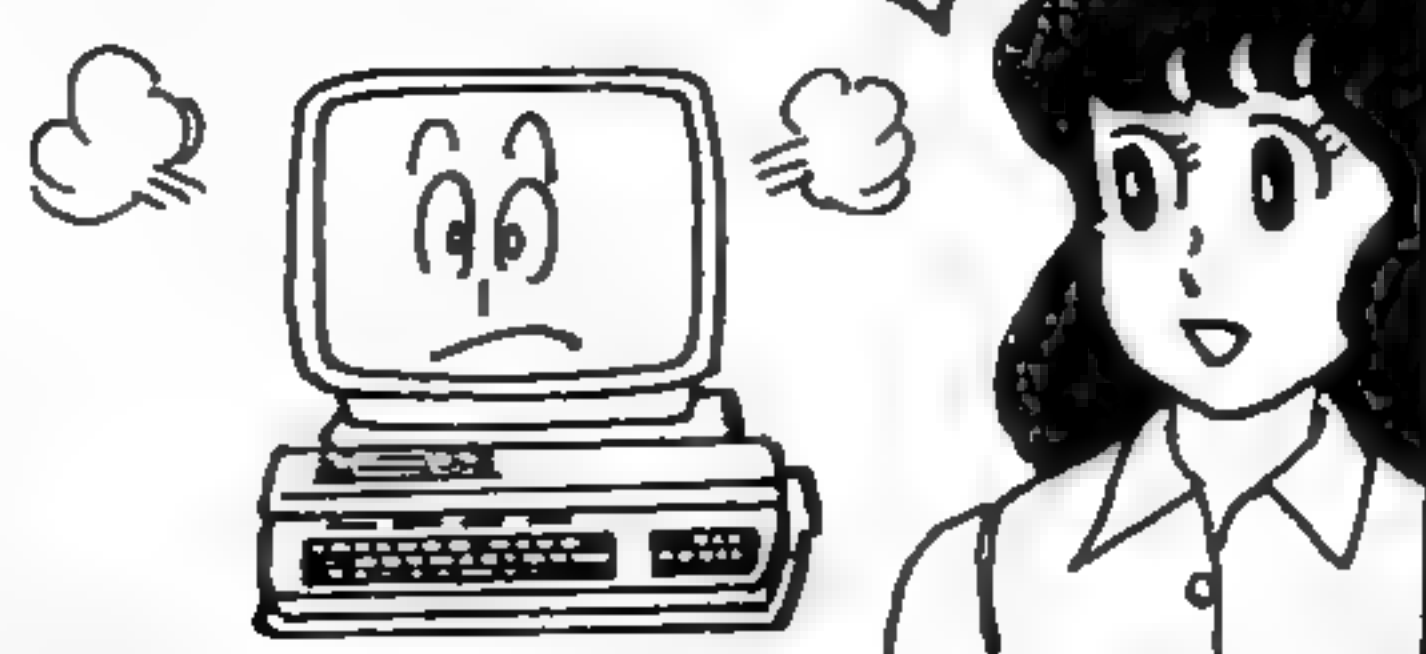
ここではたまたま手元にあったポケコンPC1500とパソコンIF800-30を使っていますが、もしFM-7やPC8001などを使ってもほとんど同様です。



ただBASIC言語と言っても現在のパソコン用に使われているものは各メーカーがいろいろ独自に開発し工夫している部分があって統一されていないということを知っておいてください。とくにRS-232Cという規格の通信ポートを使う場合などは本当にバラバラというところです。

そして何よりも実際にパソコンでキーをたたいてためていただきたいと思います。IF800など100万円近くするし、持っている人は少ないと思います。ディスクが1台ついていてCP/Mが走るパソコンであればどの機種でも同じです。

これから見ていくサンプルプログラムも多少機種による違いが出てくるかも知れませんが、そこは大きな心で読み進んでいただきたいと思います。



あるメーカーではENTER またあるメーカーでは印のキーを使っています。

OK AUTO  
10

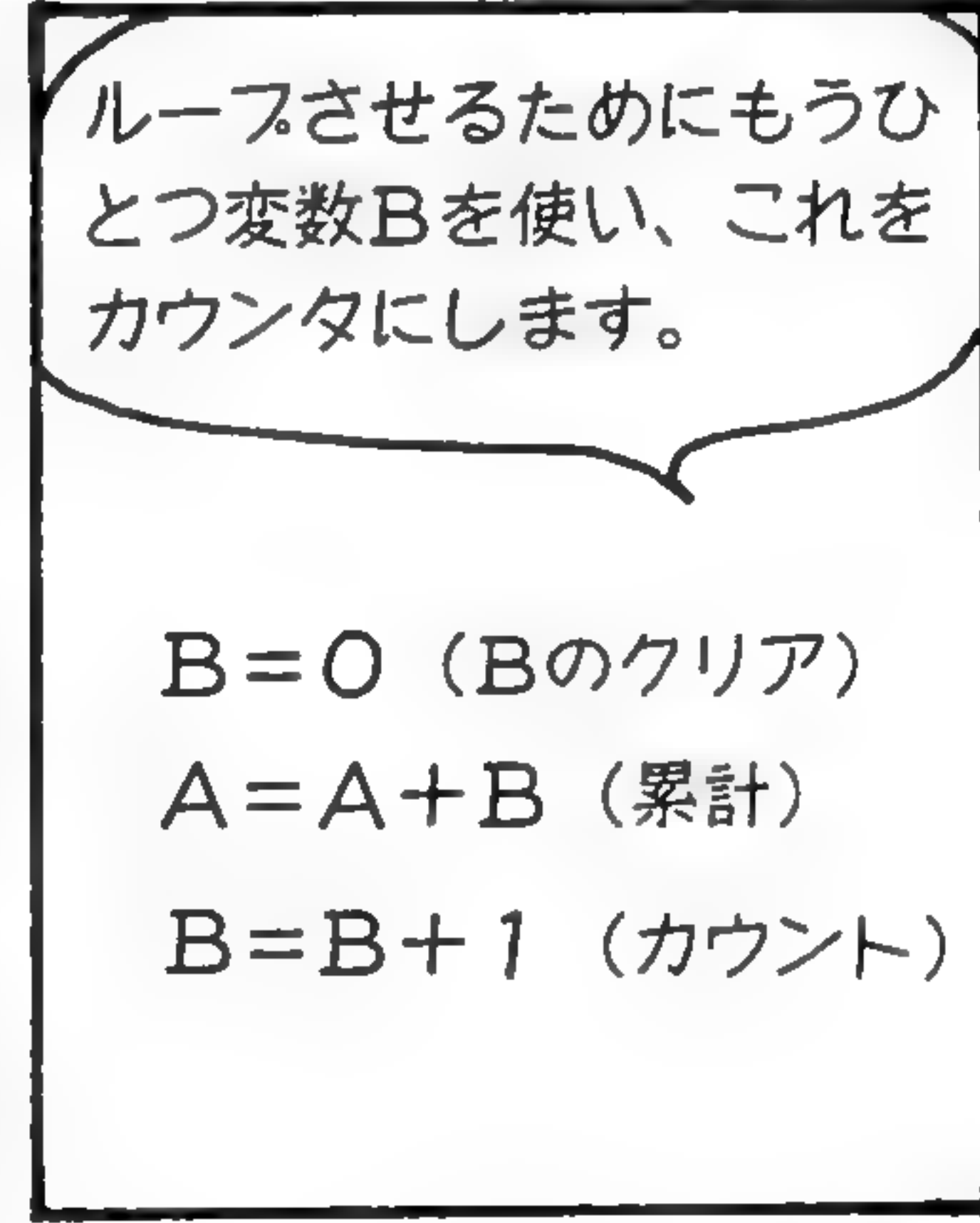
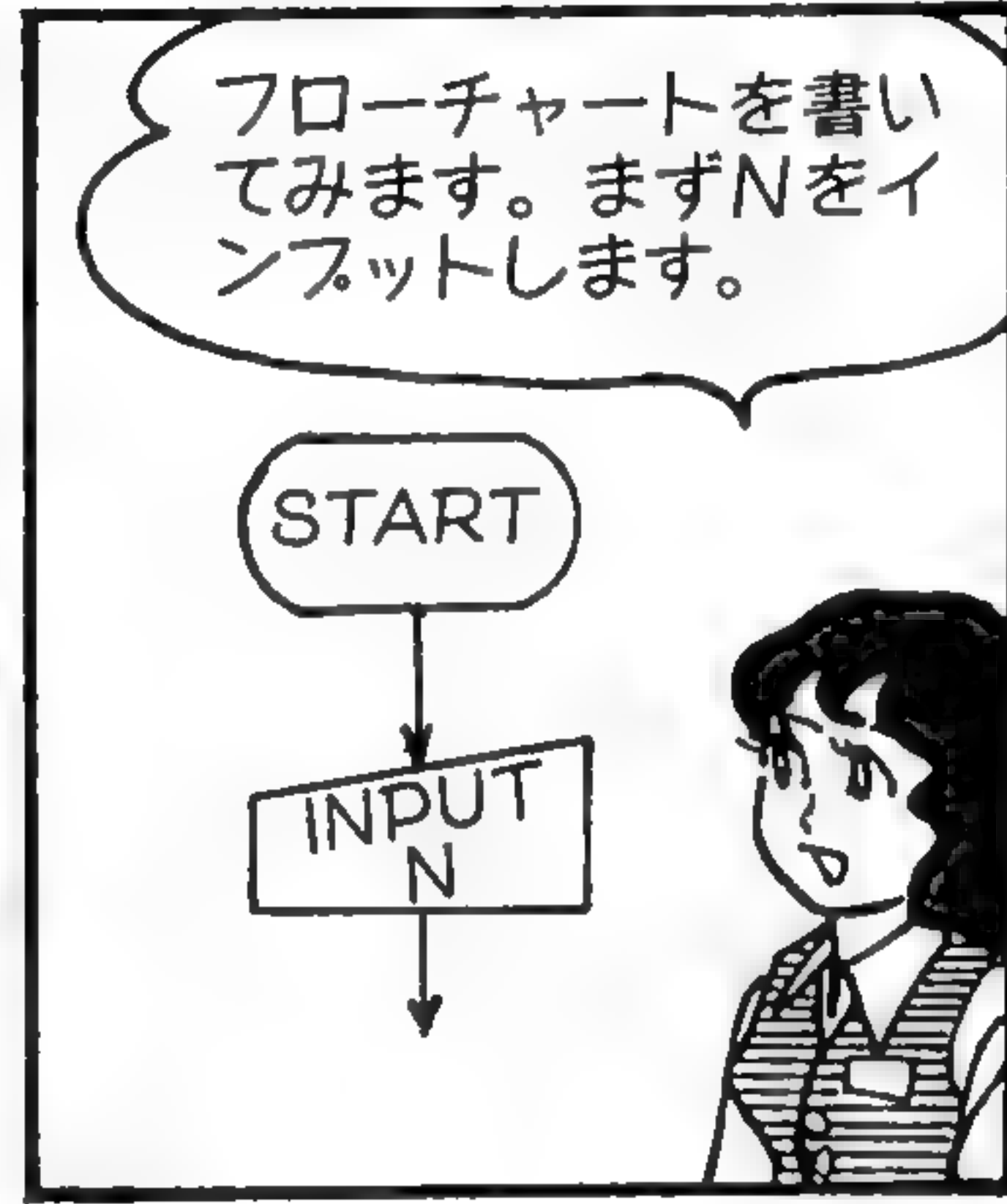
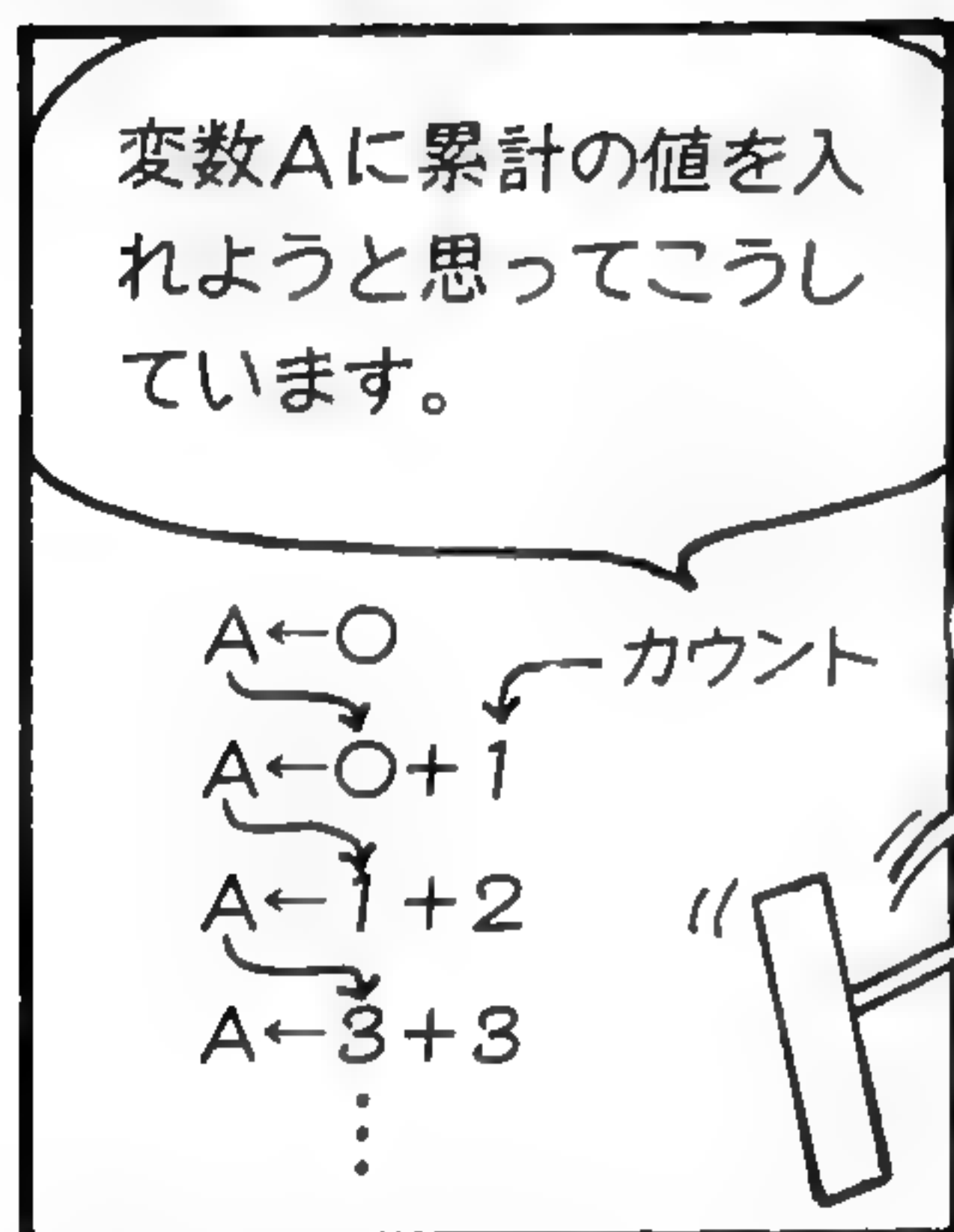
A U T O



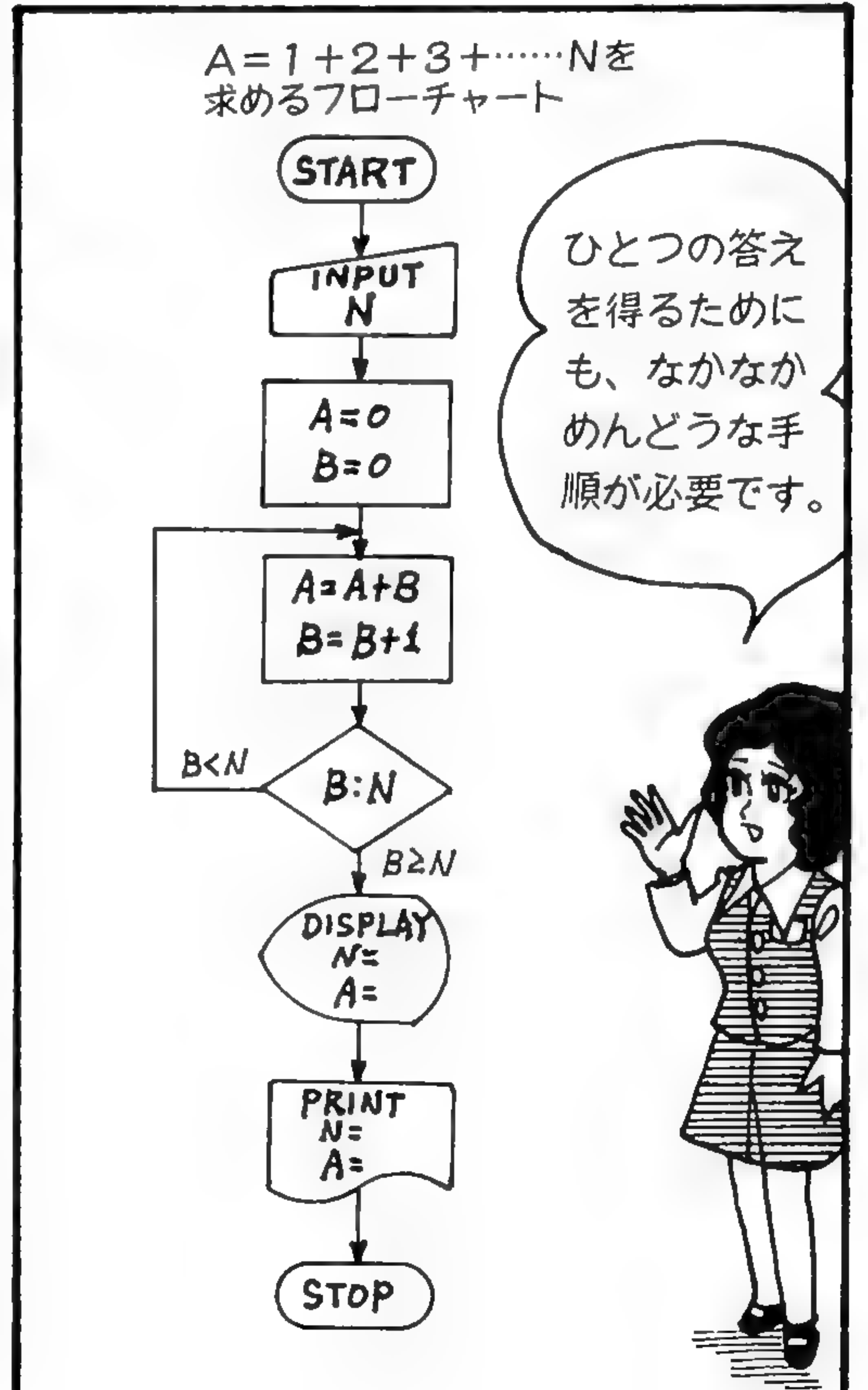
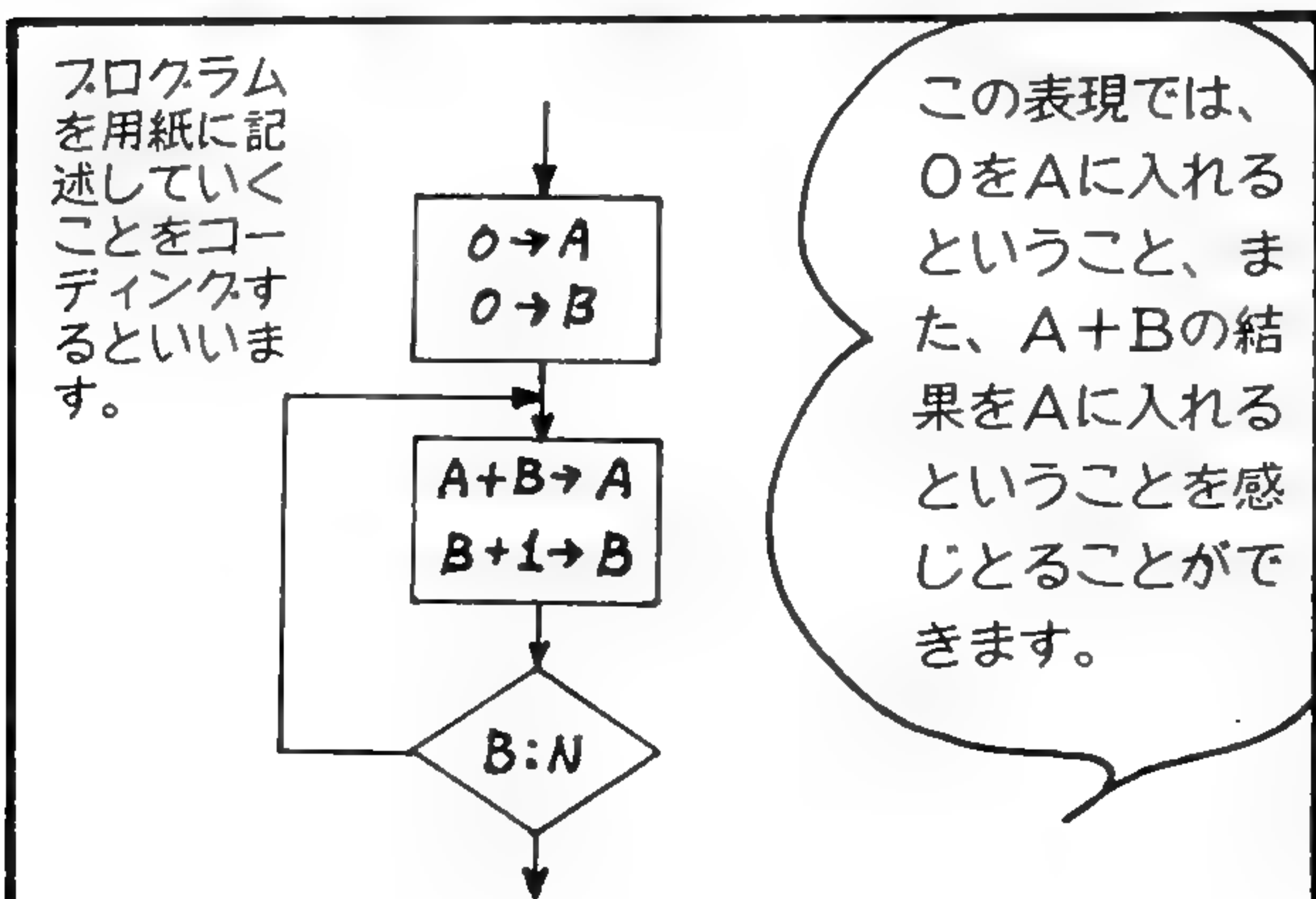
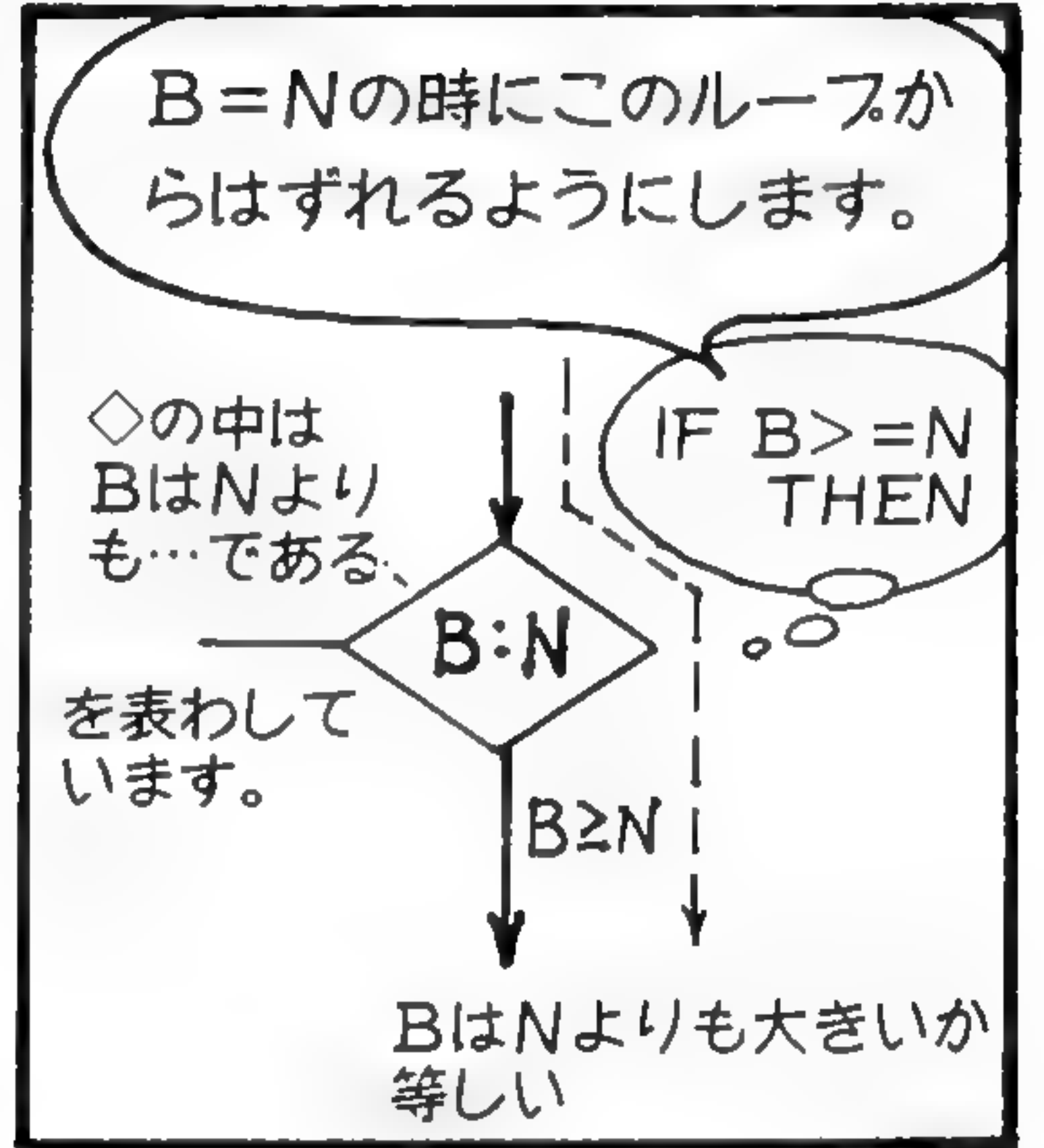
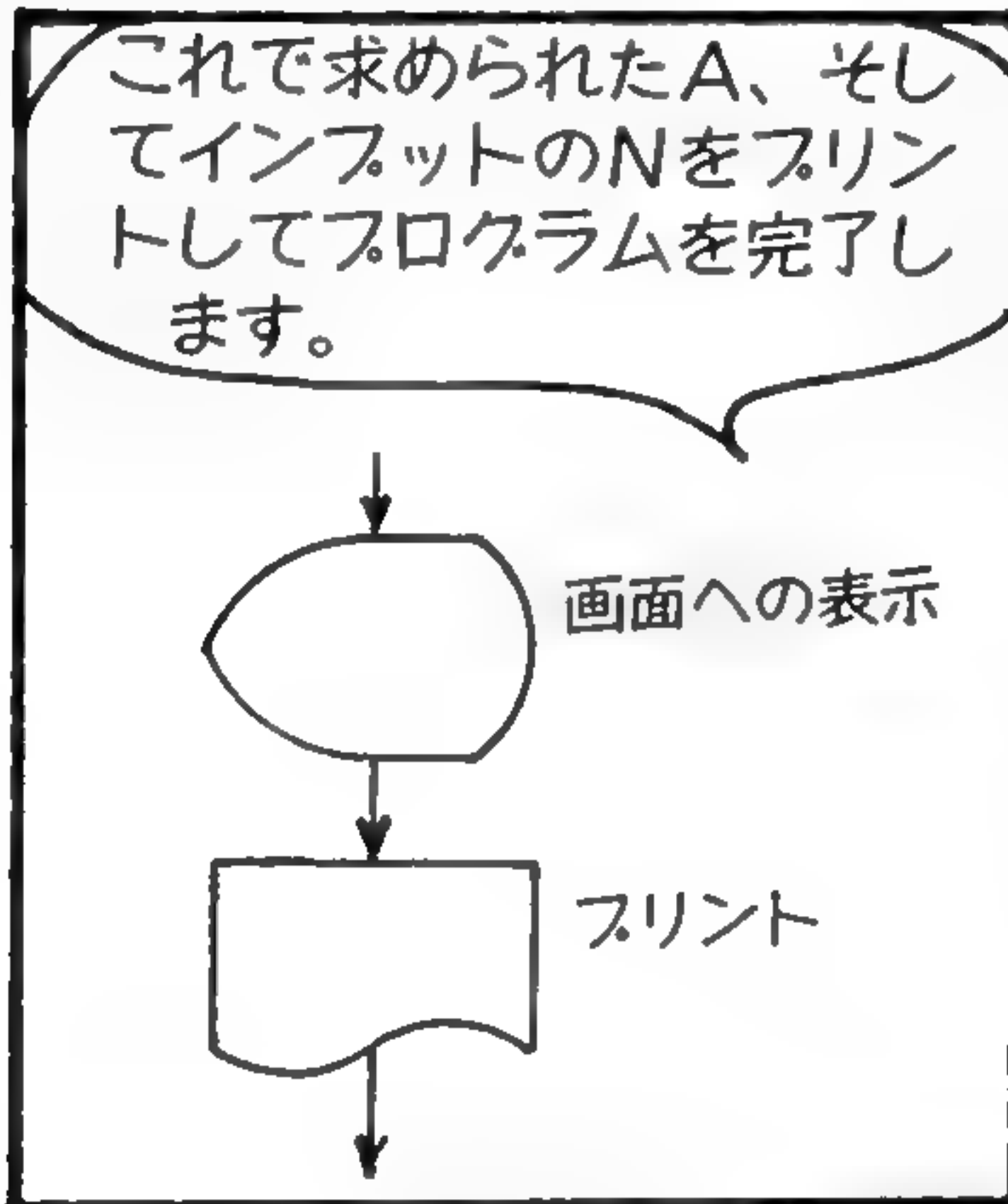
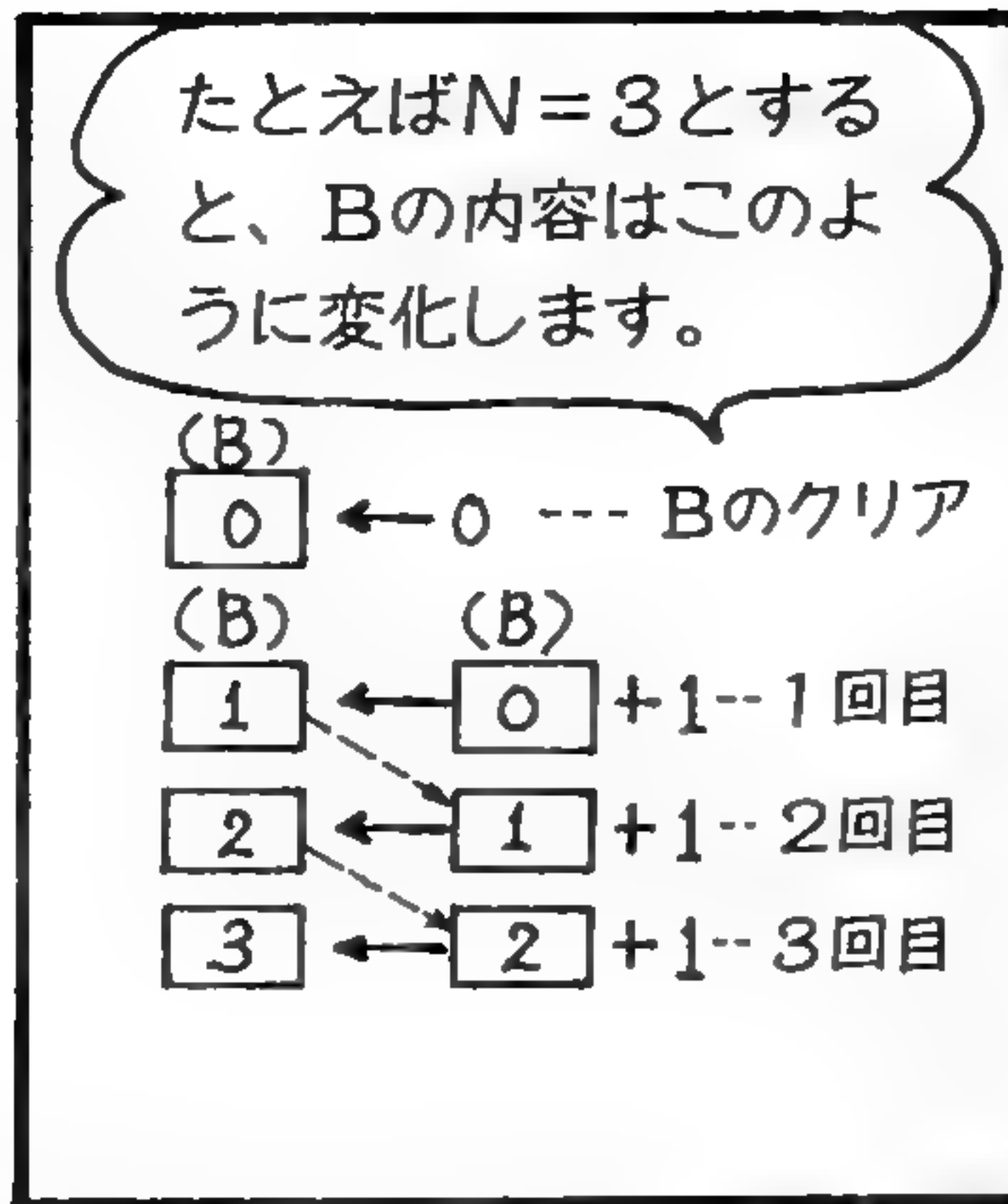
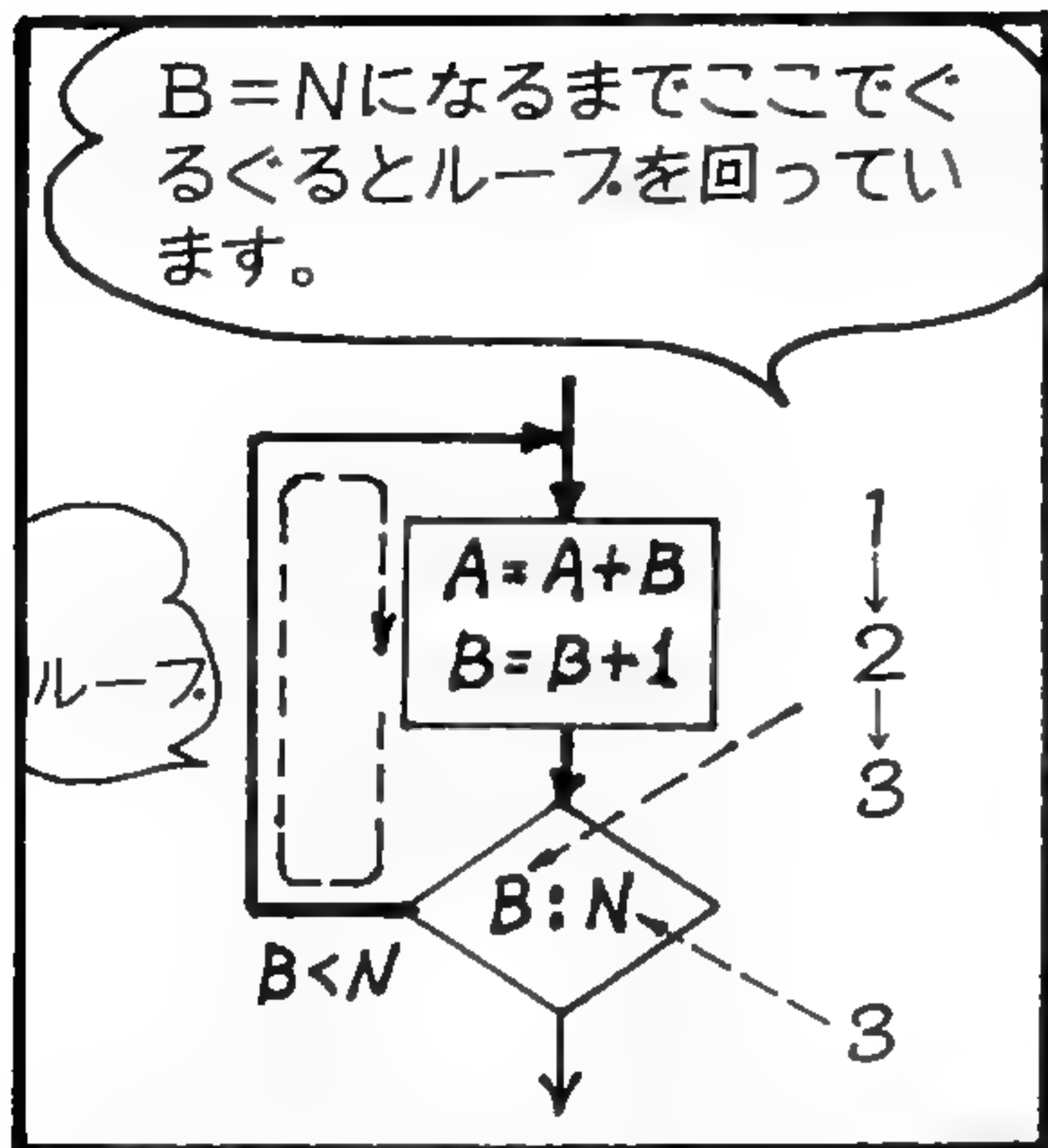


### ③ループを使って 累加計算してみよう

$$1 + 2 + 3 + \dots + N$$









これらはBASICやその他の高級言語の文法、つまり約束ごとなのですから、その原理さえわかれば、あとの数式表現でもいいと思うのです。

A=0  
B=0  
A=A+B  
B=B+1

フローチャートのこのような表現から数式に直して転記する方がわずらわしい気がします。

```

    graph TD
      A["0 → A  
0 → B"] --> B["A + B → A  
B + 1 → B"]
      B --> C[" "]
  
```

でも、私は思うのですが、あとでプログラムを書く時はA=0、A=A+Bのように書くわけですから、

A=0
B=0
A=A+B
B=B+1

このフローチャートからBASIC言語でコーディングしてみました。

```

INPUT "N=";N
A=0
B=0
A=A+B
B=B+1
IF B<N THEN GOTO [ ]
PRINT "N="; USING "####";N
PRINT "A=";A
LPRINT "N=";N
LPRINT "A=";A
END
  
```

省略できる

CRTに表示 { PRINT "N="; USING "####";N  
PRINT "A=";A  
LPRINT "N=";N  
LPRINT "A=";A  
END }

表示の桁数を決める

高級言語を使ったとしても、プログラミングし、それを完成させるまではエラーとの戦いなのです。エラーを誘うものはさけるべきです。

```

10 INPUT "N=";N
20 A=0
30 B=0
40 A=A+B
50 B=B+1
60 IF B<N THEN 40
70 PRINT "N="; USING "####";N
80 PRINT "A=";A
90 LPRINT "N=";N
100 LPRINT "A=";A
110 END
  
```

番号は適当につければよいのです。

あとで直しやすいように10とびにつけることが多いです。

この1行をステートメント(文)と言いますが、各々のステートメントに若い順に番号をつけておきます。これがBASICの特徴のひとつです。

No.	ステートメント
[ ]	INPUT "N=";N

あとは、BASICの走るパソコンにこのプログラムを入れて試してみるだけです。今手元にはPC-1500(シャープ)がありますので、これを使いましょう。

これでフローチャートとプログラムコーディングが終わりました。



**MODE** キーを押してPRO（プログラム）モードにします。AUTO 機能がないので文番号も全部キーインする必要があります。下のリストは 10: INPUT...のように文番号のあとに：がありますがこれはキーインしたものではありません。また、

60: IF B<N THEN 40 は  
60: IF B<N THEN 40 **ENTER** とキーインしたのであり、

60: IF B<N THEN 40 **ENTER** としたものではありません。キーインが完了したらもう一度 **MODE** を押してRUN（実行モード）にします。

このポケットコンピュータはROM型パソコンなので電源をONするとすぐに使えるBASICが動きます。

他のメーカーのハンドヘルドコンピュータもROM型です。



使用済バイト数確認  
STATUS 1 のためのコマンド  
**ENTER** 表示 → 121

```
10: INPUT "N="; N
20: A=0
30: B=0
40: A=A+B
50: B=B+1
60: IF B<N THEN 40
70: PRINT "N=";
  USING "####"; N
80: PRINT "A="; A
90: LPRINT "N="; N
100: LPRINT "A="; A
110: STOP
```

LISTとキーインするとプログラムのリストがプリントされます。

この1行は  
10: INPUT  
"N"; N  
**ENTER** とキーインしました。



このポケコンの場合はRAMに入れたプログラムがバッテリーでバックアップされています。

電源を切ってもプログラムが消えないのがいい点です。

DE+ PRO



**ENTER** を押すと、演算が始まり、結果が出ます。

N= 10

— — — —

10をインプットしてみましょう。10と押します。

N=10—

このプログラムをRUNさせます。まず、N=\_が表示されました。

カーソル

N= \_

RUN ENTER

BASICでは、USINGを使い、` `の中の#で形を指定します。今のプログラムをUSING `###. #` としておくと、N= 10. 0とプリントされます。

—は空白の意味です。

表示

プリント

数値をアウトプットする時、その形式を指定しておく必要があります。

この表示はN= 10と表示されているのです。これはプログラムのUSING `####` のためです。

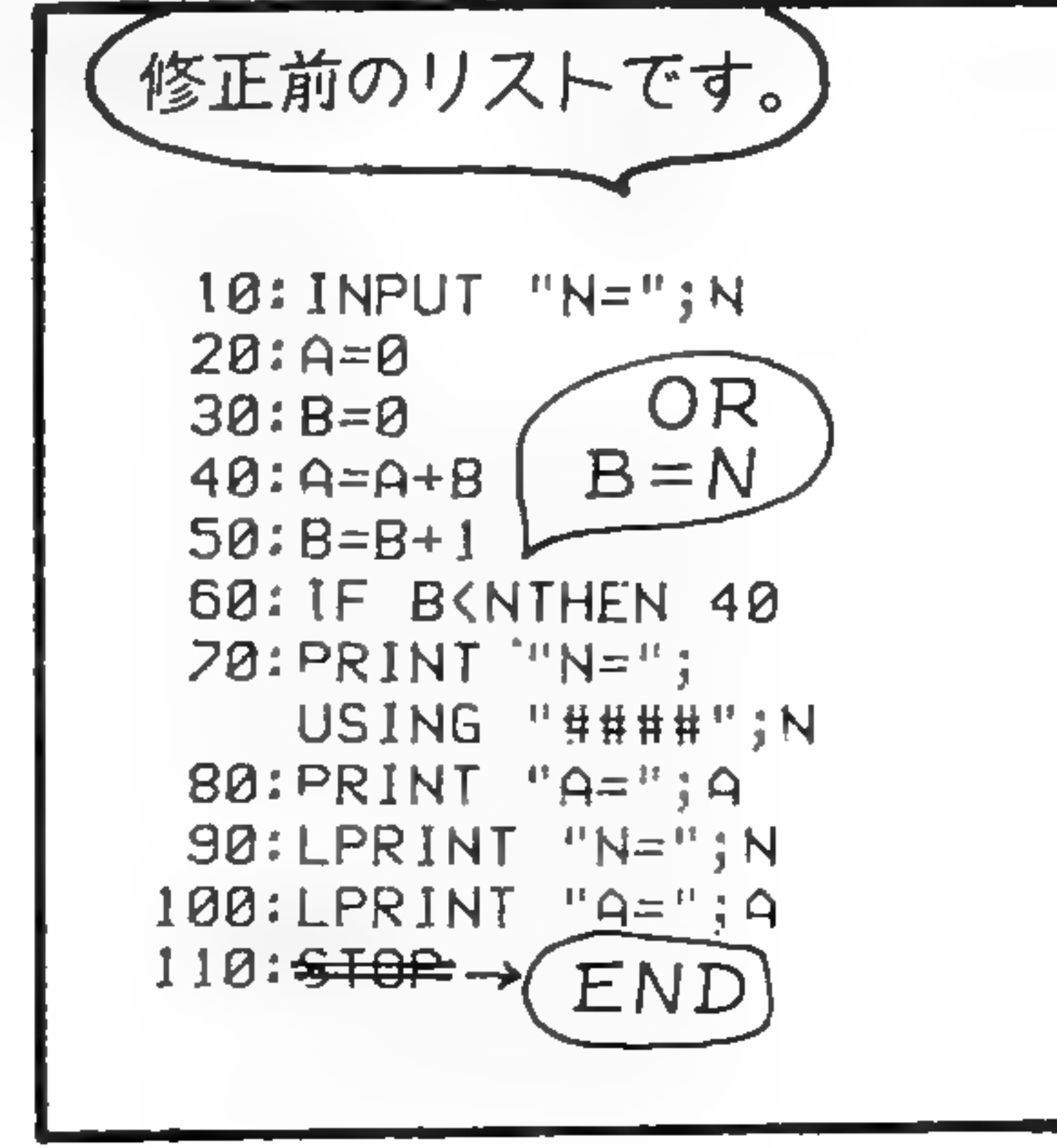
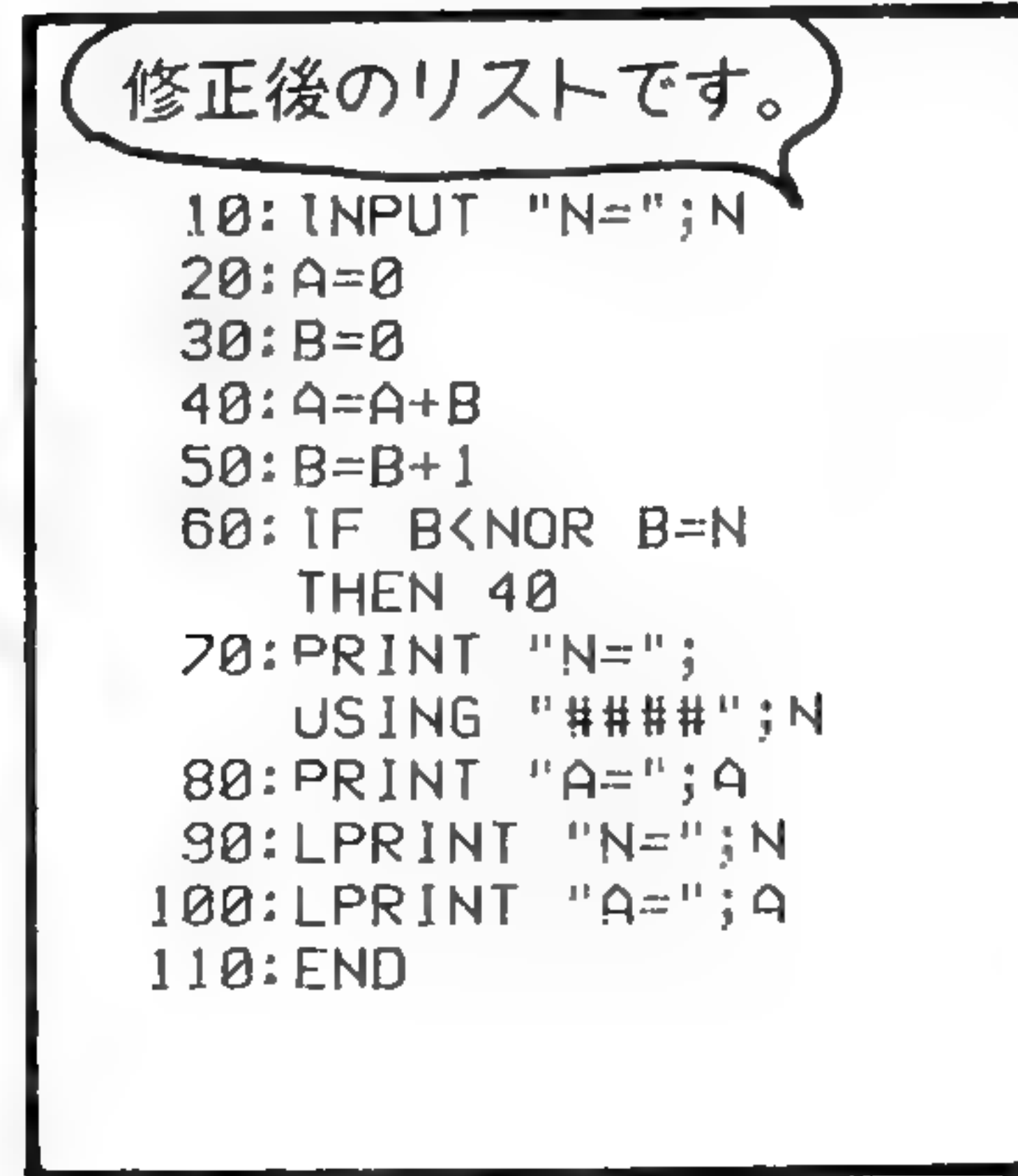
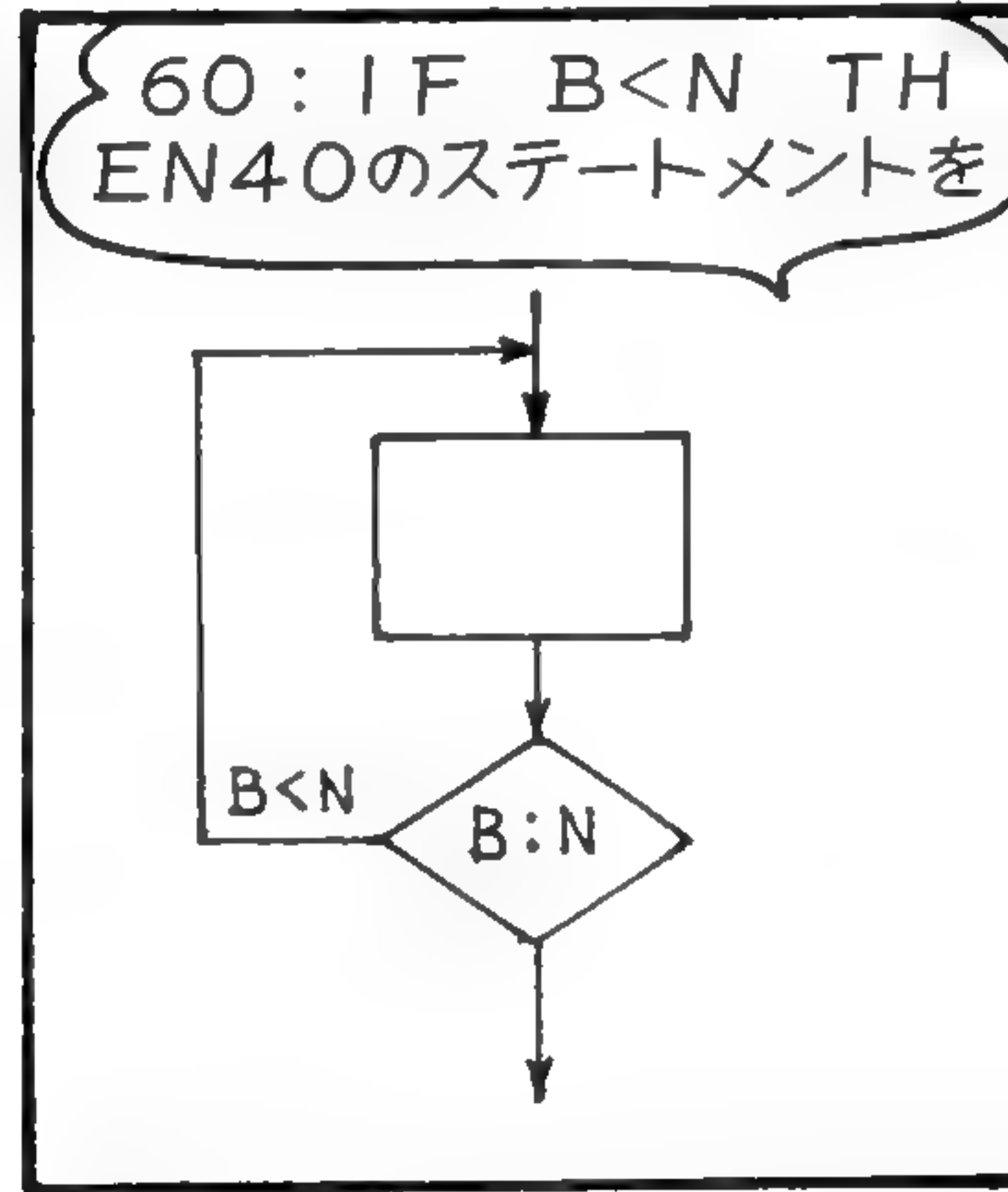
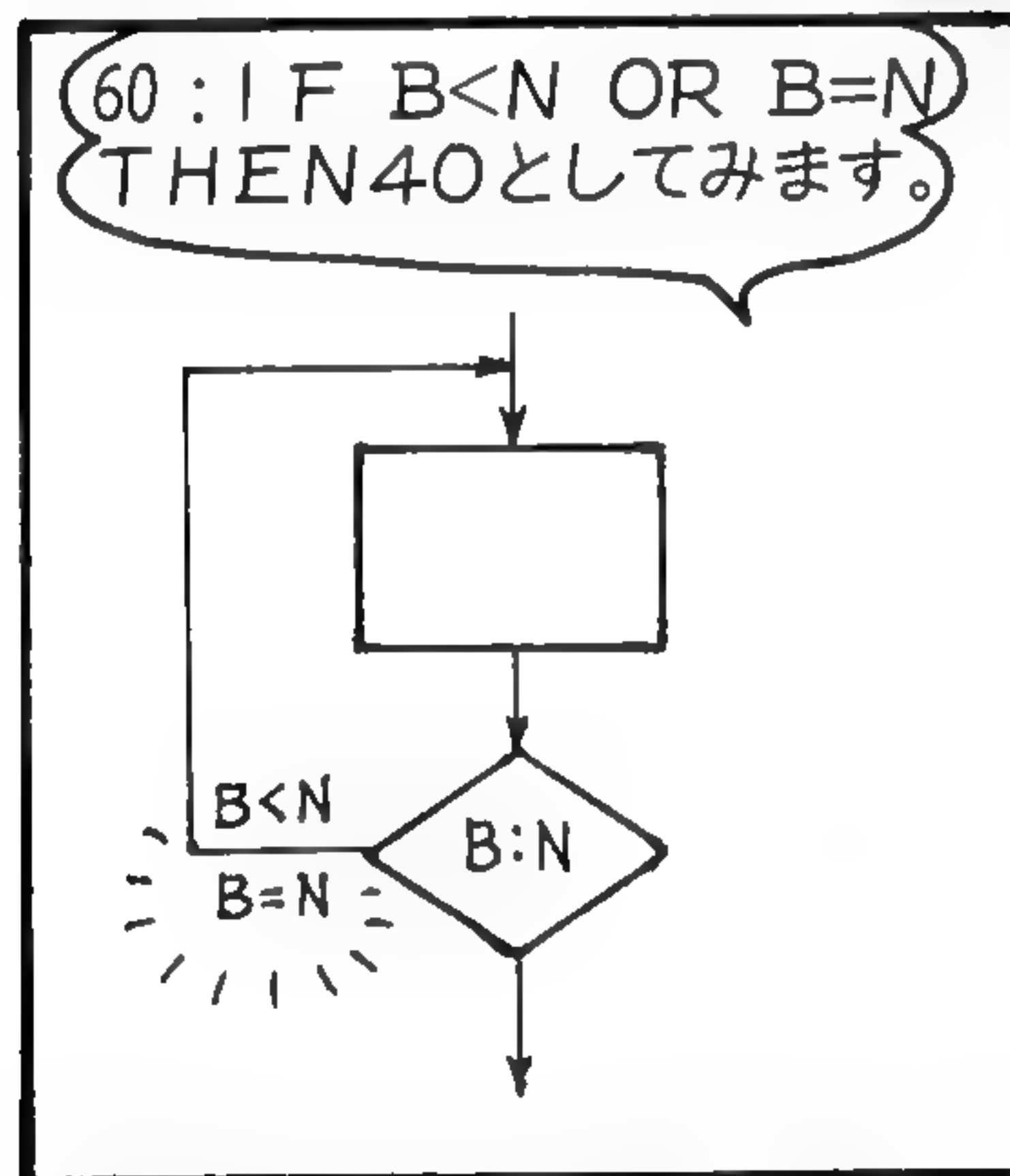
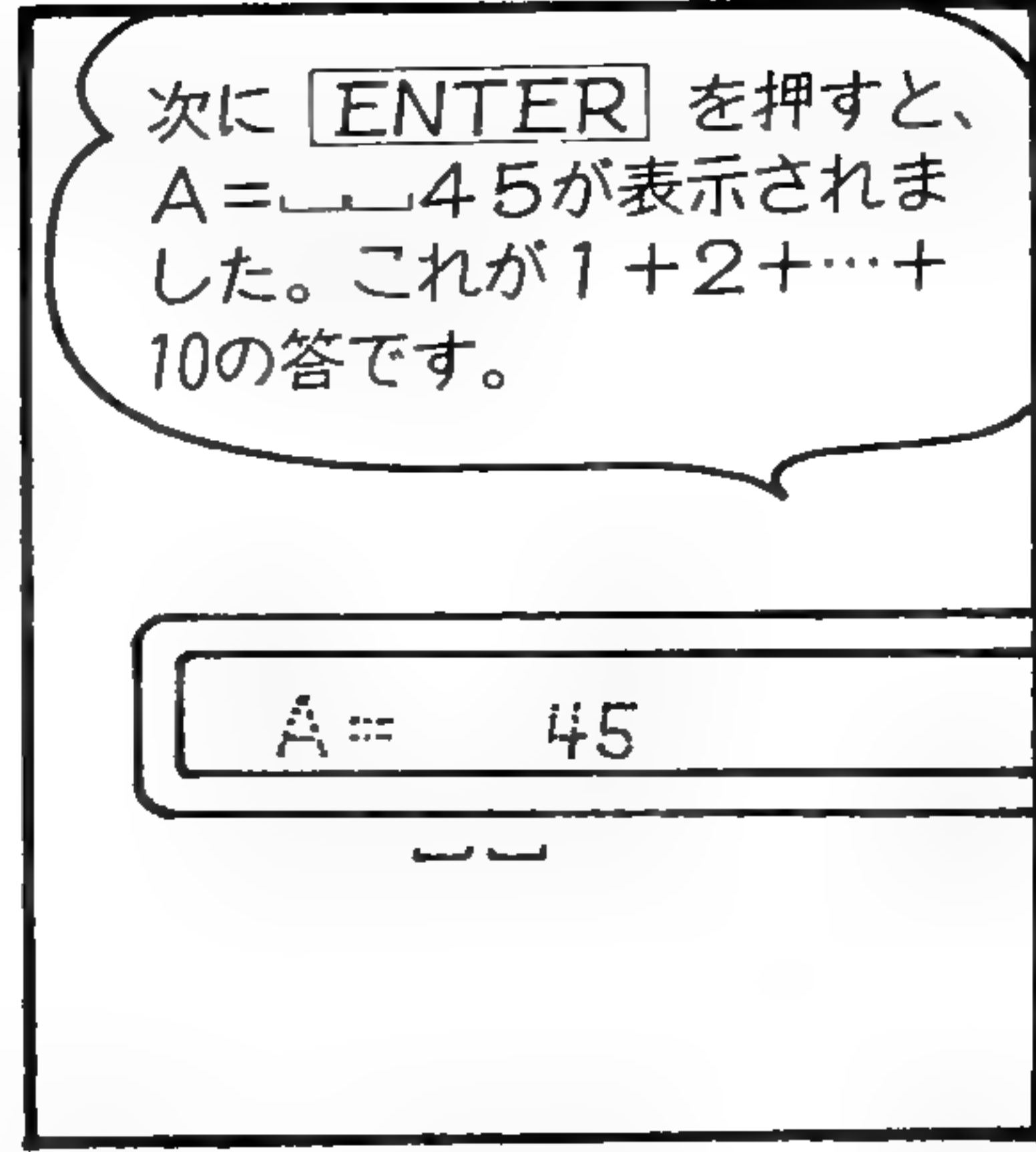
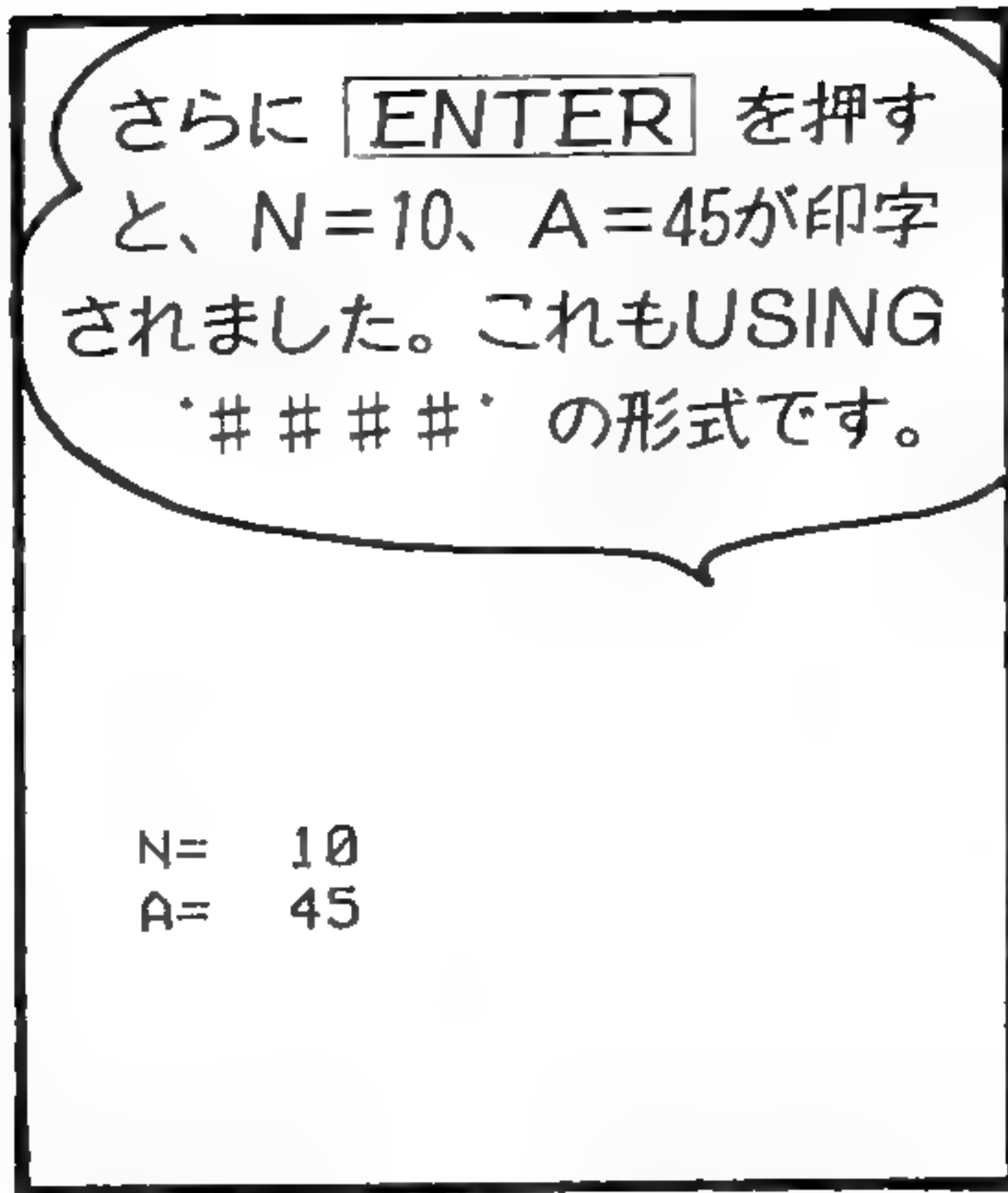
この桁は  
符号用

#####

10









プリントフォーマット(書式)の指定の仕方によっては、答が印字されない場合があります。プログラムする時、十分考慮しておく必要があります。



変更後のリストです。

```
10: INPUT "N="; N
20: A=0
30: B=0
40: A=A+B
50: B=B+1
60: IF B<NOR B=N
    THEN 40
70: PRINT "N=";
    USING "#####"; N
80: PRINT "A="; A
90: LPRINT "N="; N
100: LPRINT "A="; A
110: END
```

これはN=50ではAが4桁となるためです。そこで、USING '#####'をUSING '#####'としてみました。これだと6桁までいけます。

符号 有効数字

# #####



100回のループで5秒かかっているようです。これで任意のNの計算時間を予測できます。



演算時間

N=	50	} 2秒
A=	1275	
N=	100	} 5秒
A=	5050	
N=	500	} 25秒
A=	125250	
N=	1000	} 50秒
A=	500500	
N=1500		1分15秒
A	1125750	

(N=1500ではオーバーしたのでAの内容をマニュアルプリント)

N=50を超えると、意外と計算時間がかかるので、その時間を測ってみました。N=1000になると50秒かかりました。



手計算で1+2+...999+1000など、とても計算する気にはなれません。

1+2+3+4+5+6+7+8+9  
+10+11+12+13+14+15+16+17+18  
+19+20+21+22+23+24+25+  
26+27+28+29+30+31+32+33  
+34+35+36+37+38+39+40+41  
+42+43+44+45+46+47+48



でも、一度プログラムが完成すると事情が変わります。

N=1000では

R U N ENTER

J O O O ENTER

と押すだけです。



問題进行处理のために、プログラムすることは、根気がいることであり、手計算の方が早いこともあります。



1+2+3...+Nを求める公式がありましたね。コンピュータを使えて計算が苦にならなくなるとあまりものを考えなくなるかも知れませんね。



公式

$$1+2+3+\dots+N = \frac{N(N+1)}{2}$$

N=1000では

$$A = \frac{(N+1)}{2} = \frac{1000 \times 1001}{2} = 500500$$



## ④FOR～NEXT文を使いこなそう

```
40 A=A+B
50 B=B+1
60 IF B<=NTH
EN40
```

はFOR～NEXT文を使う  
とこのように書けます。

```
FOR B=1 TO N
A=A+B
NEXT B
```



これを使えば、簡単にループルーチンを実現できます。



```
10: INPUT "N="; N
20: A=0
30: FOR B=1 TO N
40: A=A+B
50: NEXT B
60: PRINT "N=";
  USING "#####"; N
70: PRINT "A="; A
80: LPRINT "N="; N
90: LPRINT "A="; A
100: END
```

これを使って1+2+3+...+Nの  
プログラムを書き直してみました。



普通IとかJの  
変数を使う。

1からNまで1ずつB  
を変化させるという意味。

FOR B=1 TO N STEP 1

{この間の文の中にBがあれば1ずつ  
変化する。

NEXT B

STEP 1は省略  
できる。

同じ内容のプログラムでも、  
FOR～NEXT文を使った  
方が半分ぐらいに短縮されて  
います。



演算時間

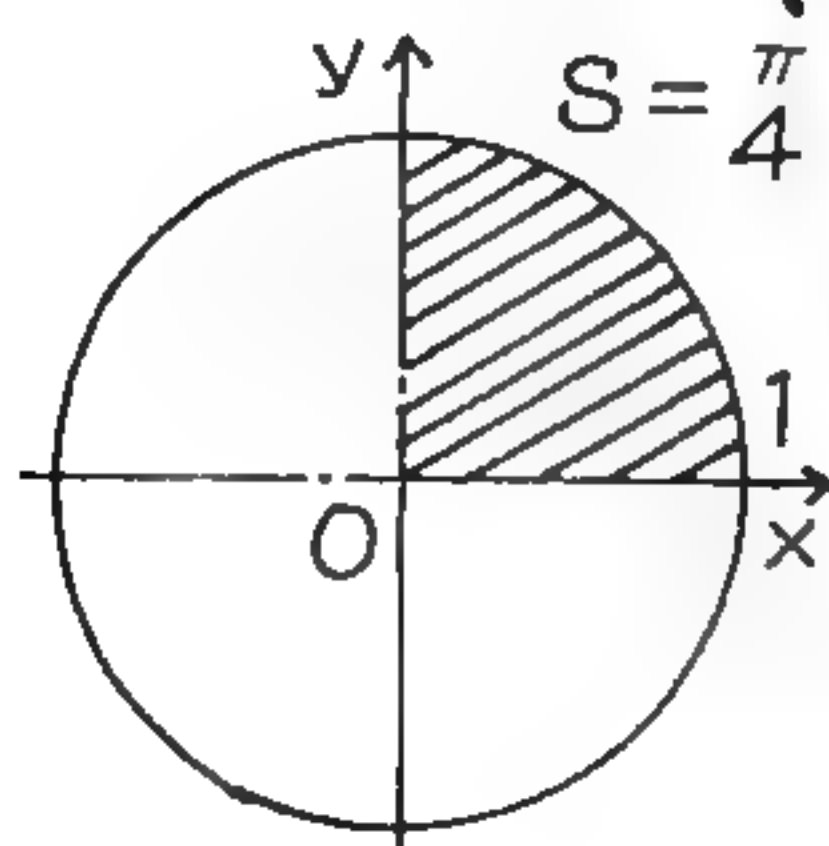
N=	50	}	1.6秒
A=	1275		
N=	100	}	2.9秒
A=	5050		
N=	500	}	13.2秒
A=	125250		
N=	1000	}	26.6秒
A=	500500		

さうそく、手元のポケコ  
ンでテストしました。

ついでに演  
算時間も測  
りました。



半径1の円の面積は  
 $\pi$ ですね。  
これの $\frac{1}{4}$ の面積を求め、あと  
で4倍すればいいでしょう。



ではここで、ひとつ別の問  
題をやってみましょう。円  
周率 $\pi$ を求めてみたいと思  
います。

$\pi = 3.14 \dots$

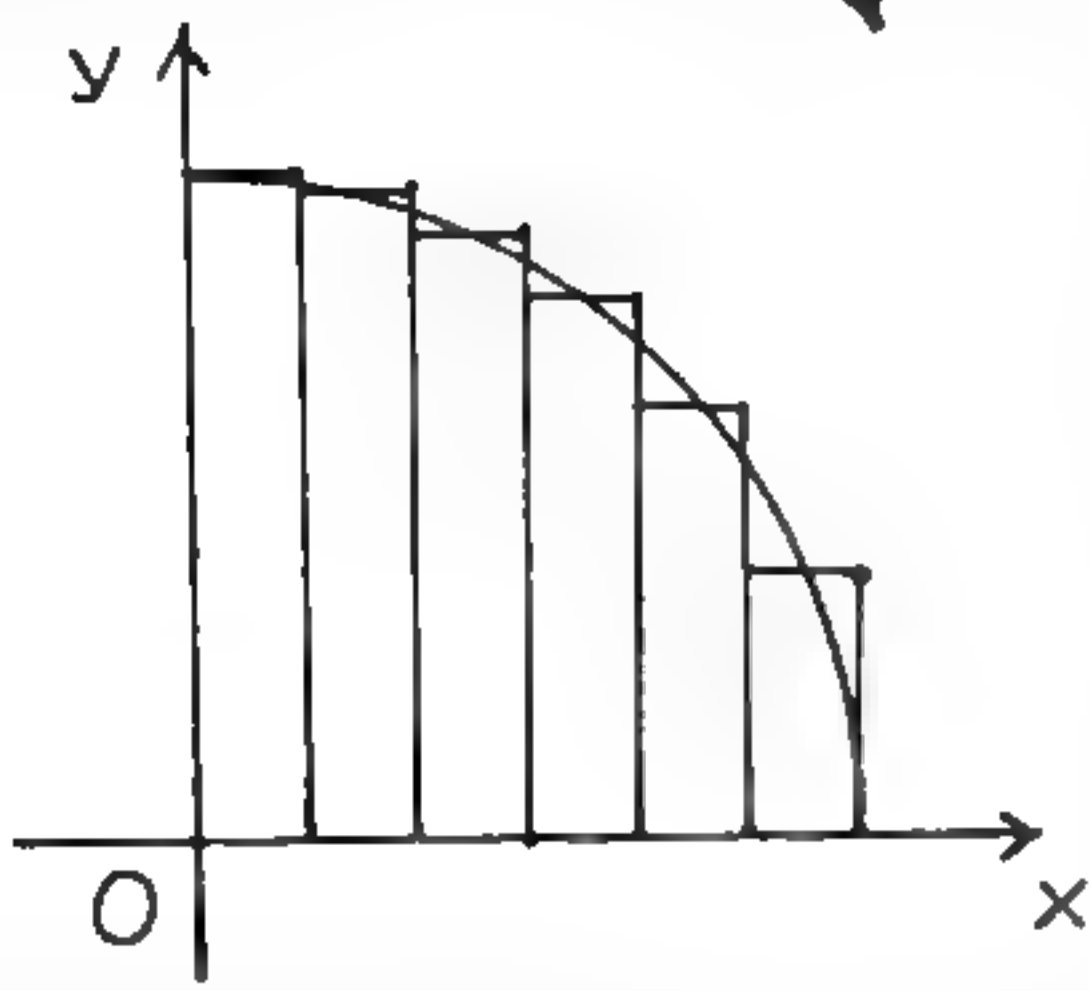


繰返し演算のある場合、プロ  
グラムの作り方によって処理  
時間が大きく開くことがあり  
ます。

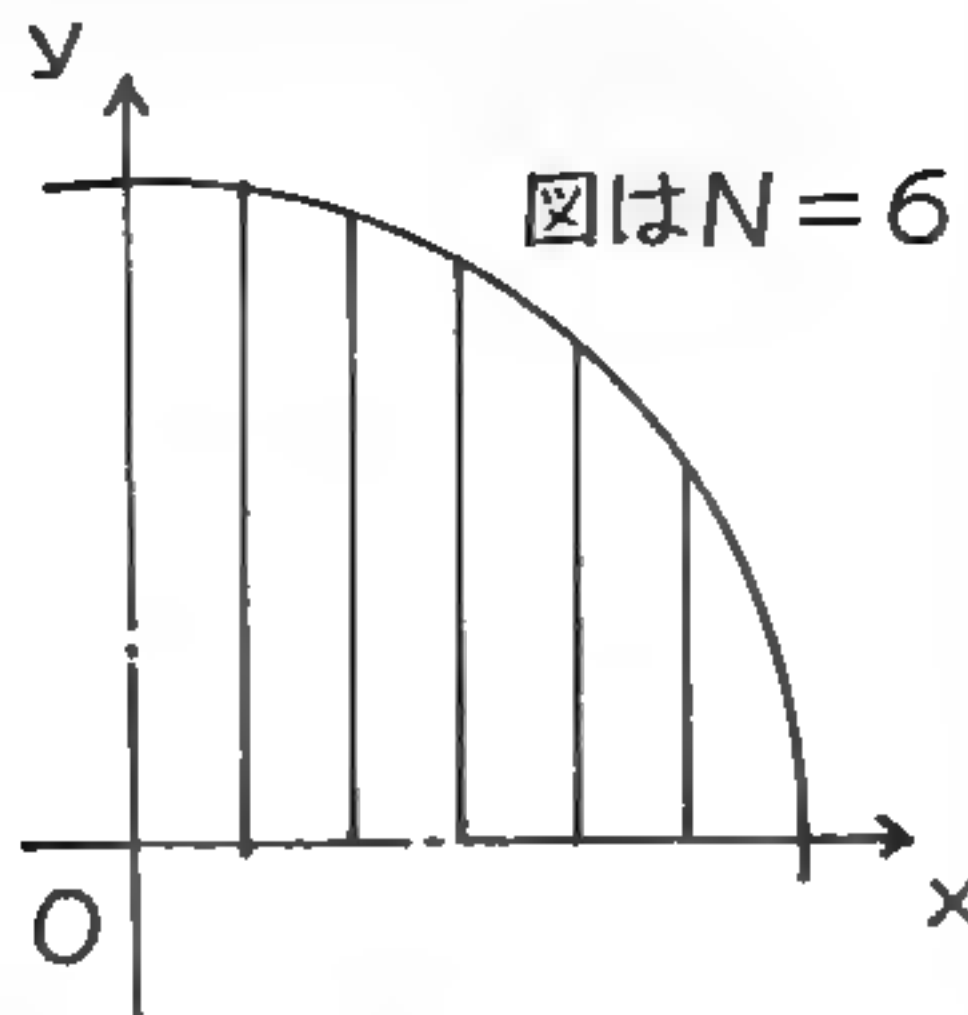




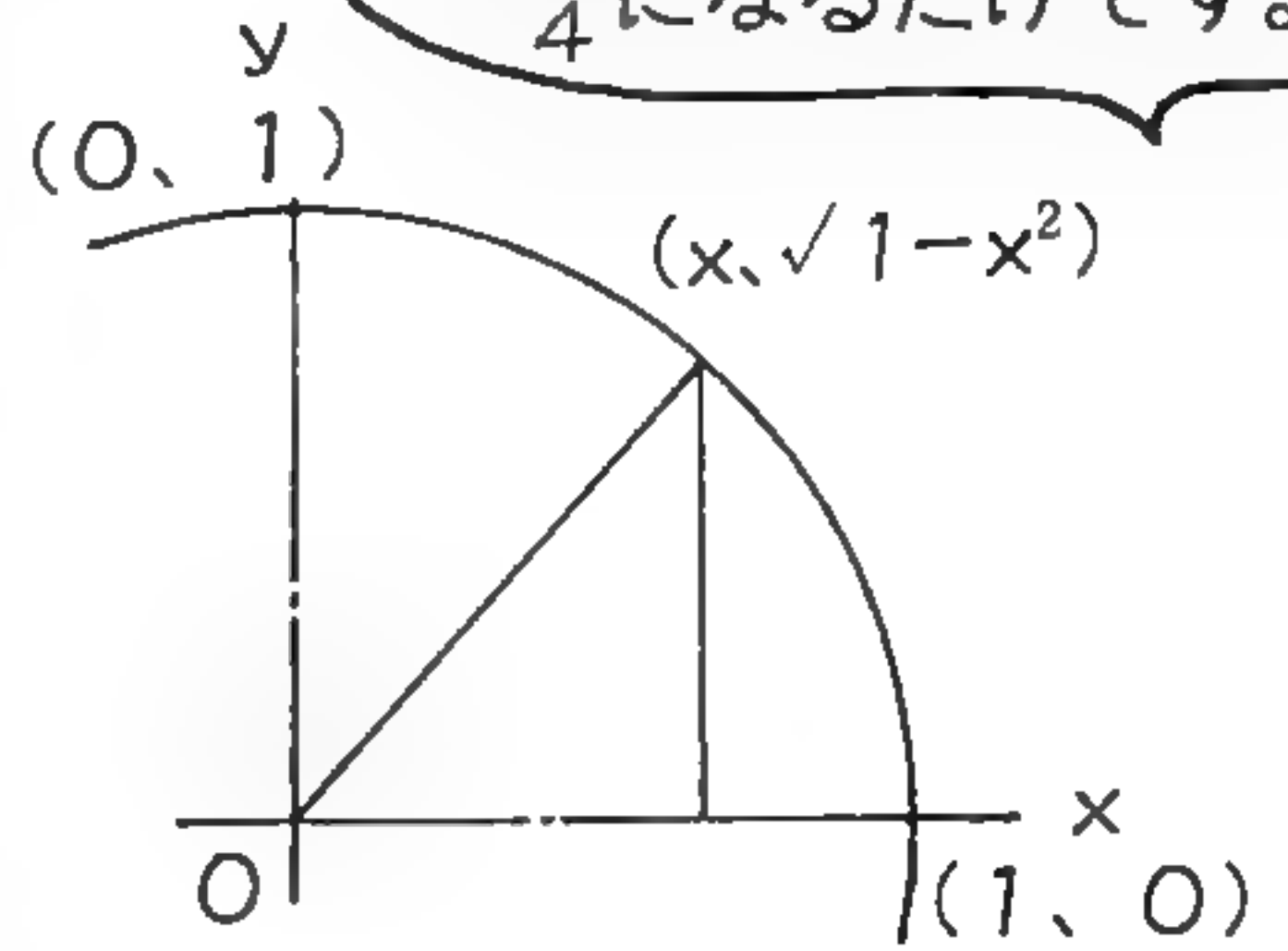
このN分割されたそれぞれの長方形の面積を合計すると、その値は $\frac{\pi}{4}$ の近似値になります。



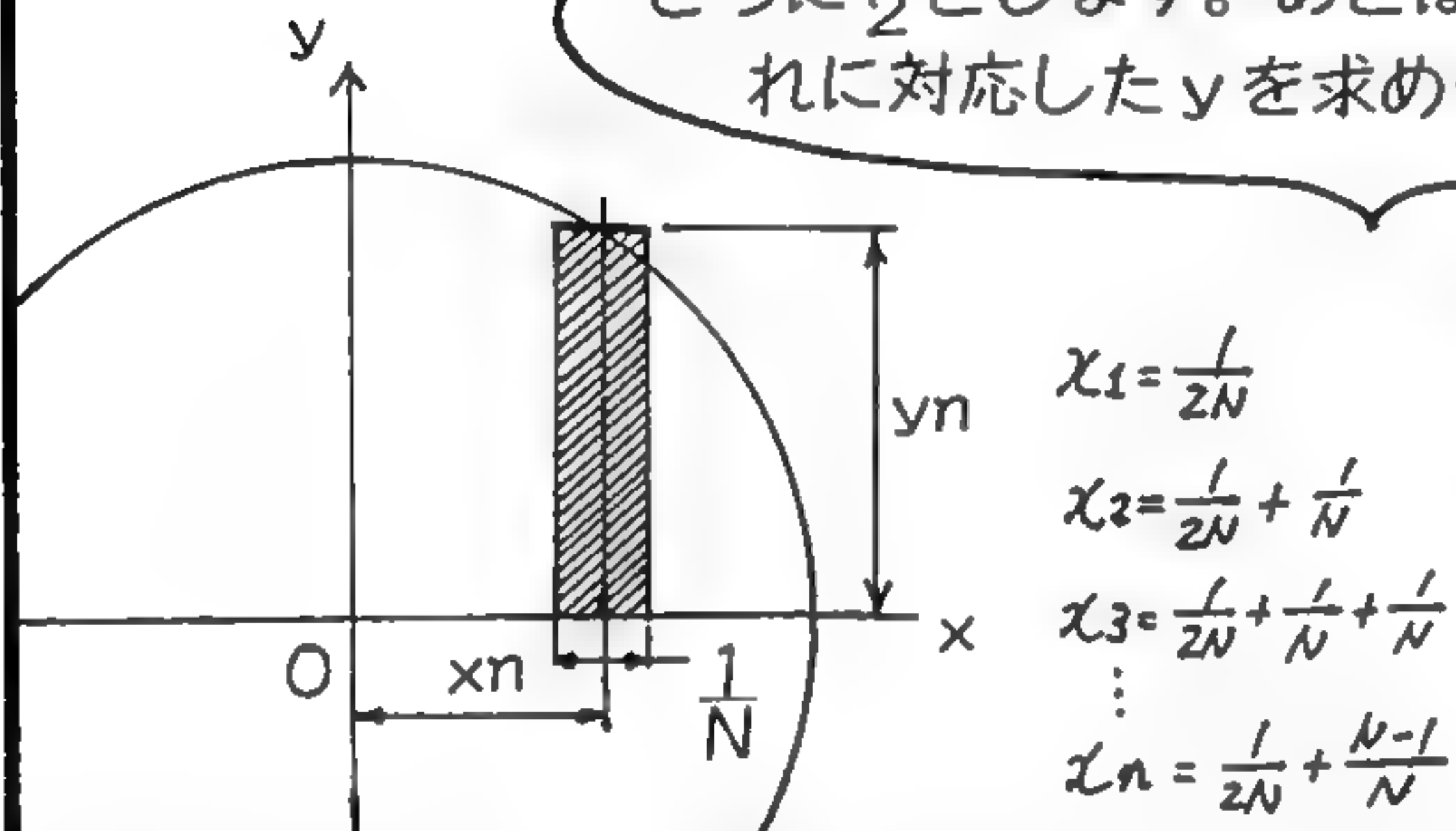
コンピュータがあると、こんな求め方ができます。 $x$ をN分割します。



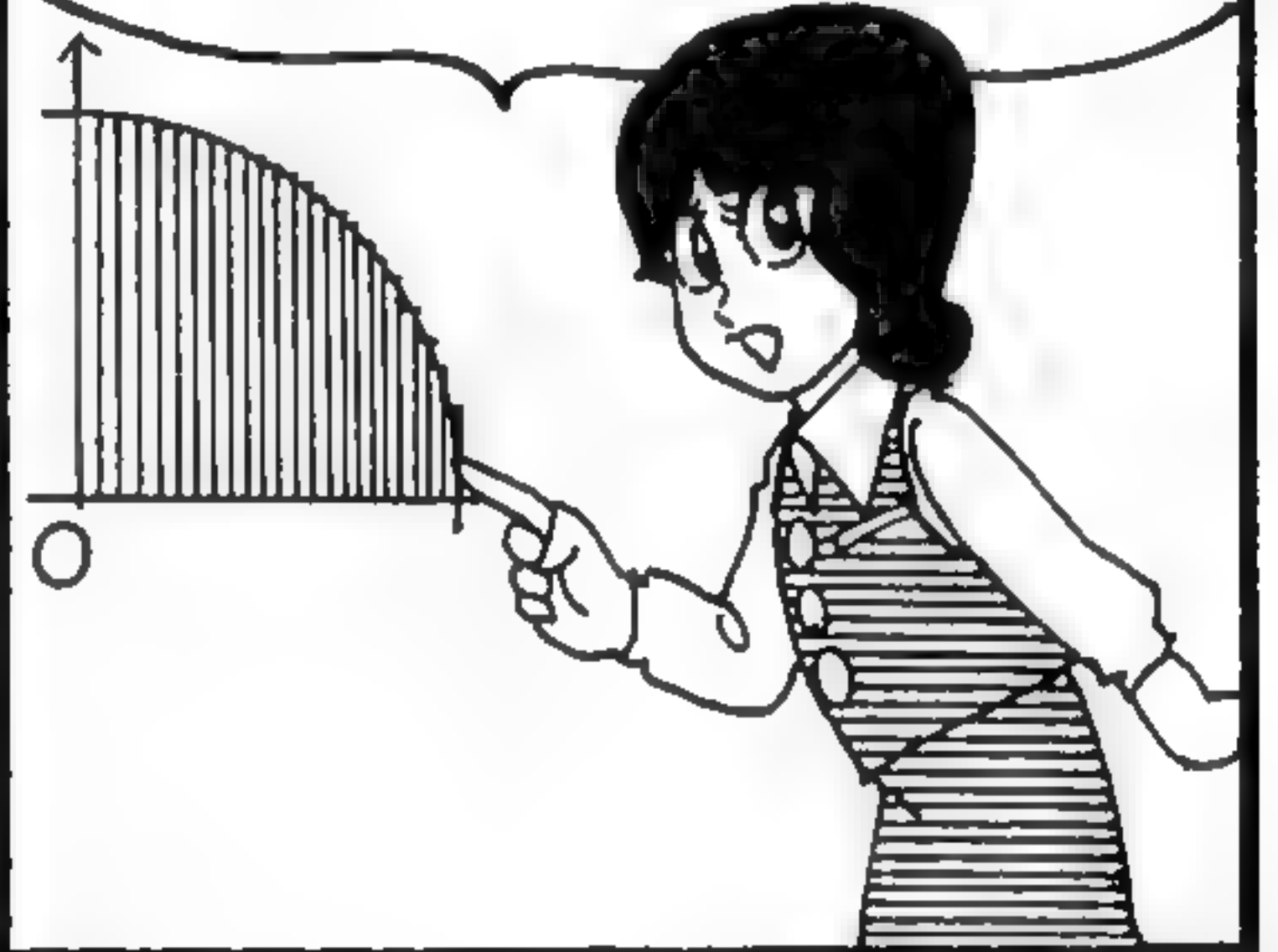
$\pi$ の値を使わずに、この面積を求めるのはむずかしいですね。積分しても $\frac{\pi}{4}$ になるだけです。



プログラムするために、計算方法を決めなければなりません。まず初期値ですが、区分 $\frac{1}{N}$ のさらに $\frac{1}{2}$ とします。あとは $\frac{1}{N}$ ずつ増加させそれに対応した $y$ を求めます。



この分割を十分大きくすると、かなり $\pi$ に近い値が得られるはずです。



#### $\pi$ の近似値をFOR~NEXTで

```

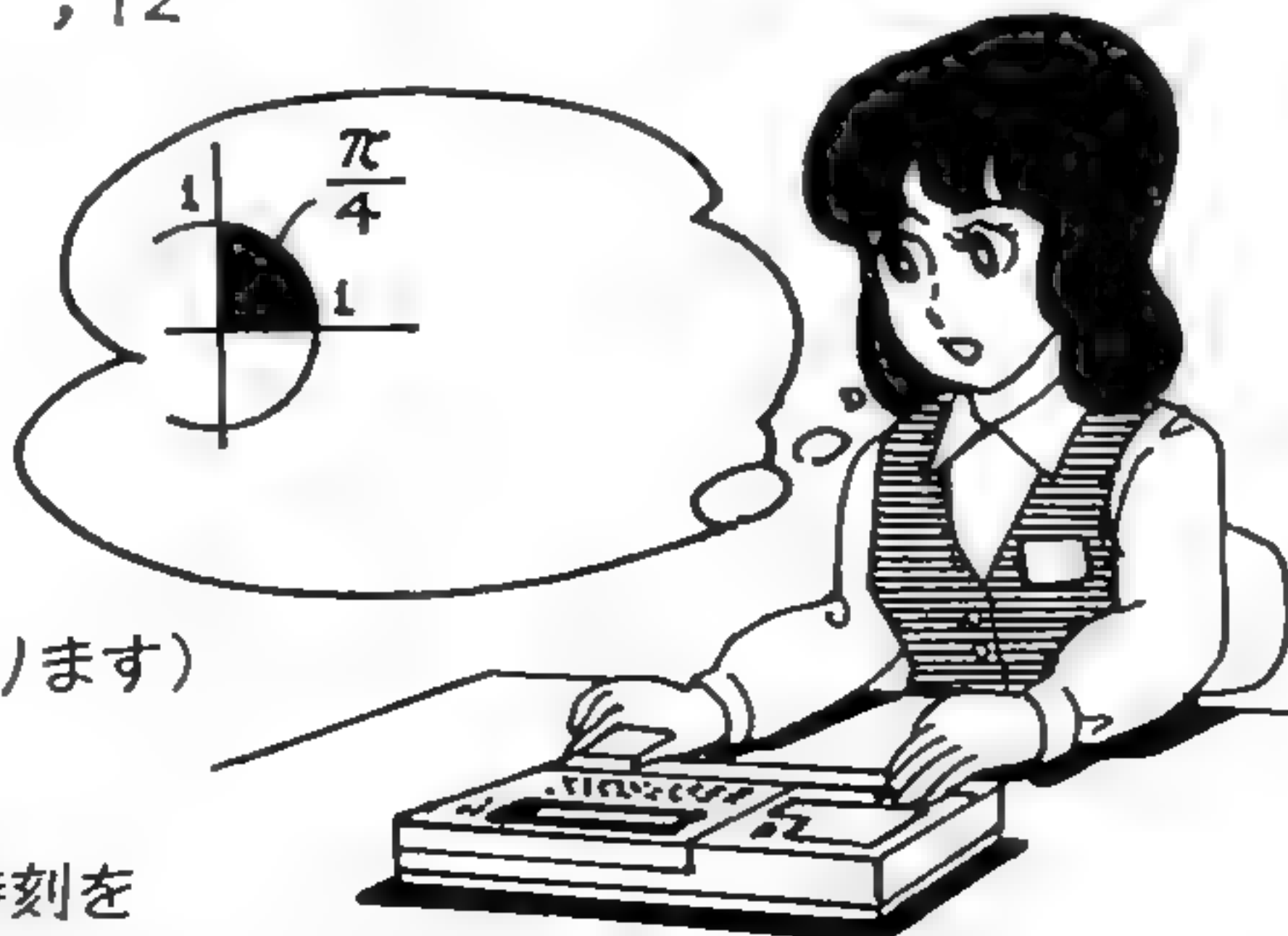
10 WAIT 0
20 INPUT "N="; N
30 T1=TIME
40 B=0:C=1/N/2:D=1/N
50 FOR I=1 TO N
60 A=D*sqrt(1-C*C)
70 B=B+A
80 C=C+D
90 PRINT USING "#####"; I
100 NEXT I
110 B=4*B
120 T2=TIME
130 LPRINT "N="; USING "#####"; N
140 LPRINT "π="; USING "#####"; B
150 LPRINT "S.TIME="; USING "#####"; T1
160 LPRINT "E.TIME="; T2
170 LF1
180 GOTO 20
190 END

```

#### 変数の意味

A... $\Delta S$   
 B...S(文110で $\pi$ になります)  
 C... $x$

TIMEはポケコンの現在時刻を得るための関数です。プリンタがついていない時は文130のLPRINTでエラーになります。



長方形の面積は、区分 $\frac{1}{N}$ と高さ $yn = \sqrt{1-xn^2}$ から求められます。

$$\Delta S_n = \frac{1}{N} \cdot y_n$$

$$= \frac{1}{N} \cdot \sqrt{1-x_n^2}$$

$$S = \Delta S_1 + \Delta S_2 + \Delta S_3 + \dots + \Delta S_n \approx \frac{\pi}{4}$$


求められたSを4倍すれば、 $\pi$ の近似値が得られます。次のコマの文番号40から110までが $\pi$ を求めるためのルーチンです。このプログラムはポケコンPC-1500のBASIC言語を使っています。





Nの最大値32767までやってみました。

N= 5000  
 $\pi = 3.141593628$   
 N= 10000  
 $\pi = 3.141593008$   
 N= 15000  
 $\pi = 3.141591457$   
 N= 19000  
 $\pi = 3.141591691$   
 N= 20000  
 $\pi = 3.141592793$   
 N= 32767  
 $\pi = 3.141593548$   
 PC-1500の $\pi$ の値は、  
 3.141592654です。



分割を多くすると、時間もかかります。N=10,000では、32分57秒かかっています。もちろんこれはPC-1500のスピードです。



ためにN=10としてやってみると $\pi = 3.1524\dots$ と出ましたので、分割を多くして、やってみました。



BASICでの通常の有効桁は6桁、倍精度で12桁です。



細かく分割すればするだけ精度が上がるといってもなさそうです。




これを見ると、信頼できるのは、3.14159までです。



コンピュータの最小単位は0か1かの1ビット  
 小数点以下の計算は二ガ手なのです。  
 小数点以下のケタ数が多いと、累計した時も誤差が大きくなります。

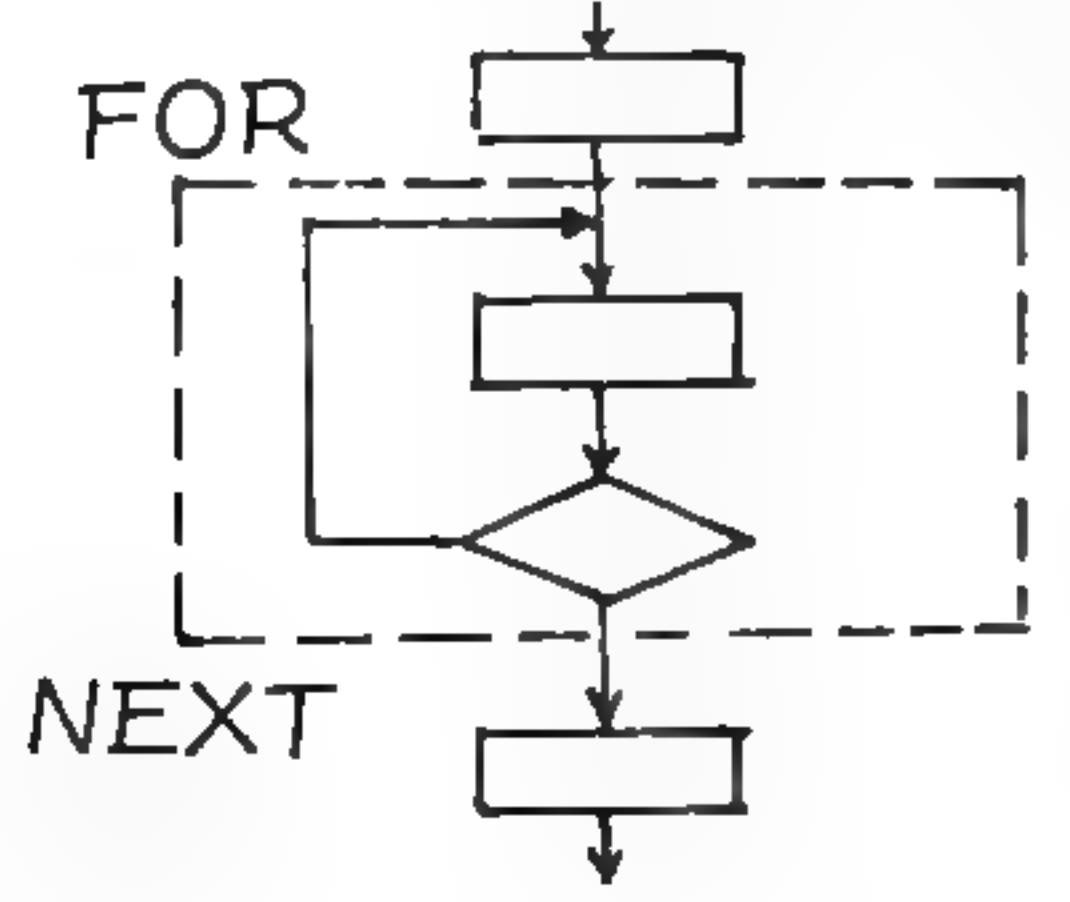

2進	10進
0.1	..... 0.5
0.01	..... 0.25
0.001	..... 0.125
0.0001	..... 0.0625



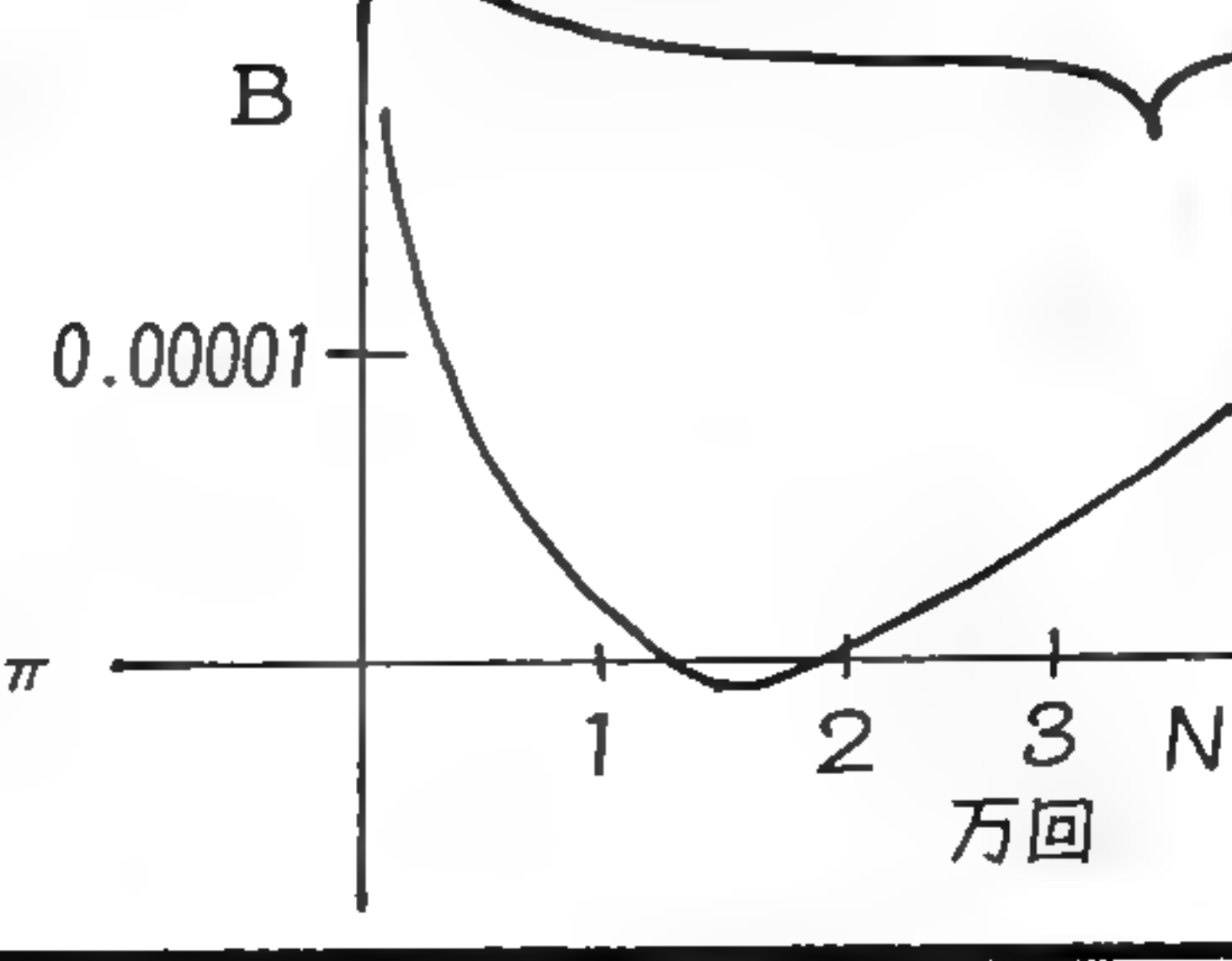

この計算では、平方根も入っていますから、累積誤差も大きくなる可能性があります。



FOR~NEXT構文は、使い慣れてくるととても便利です。同じ構文は文こそ違え、FORTRANその他のコンパイラ言語にもあります。

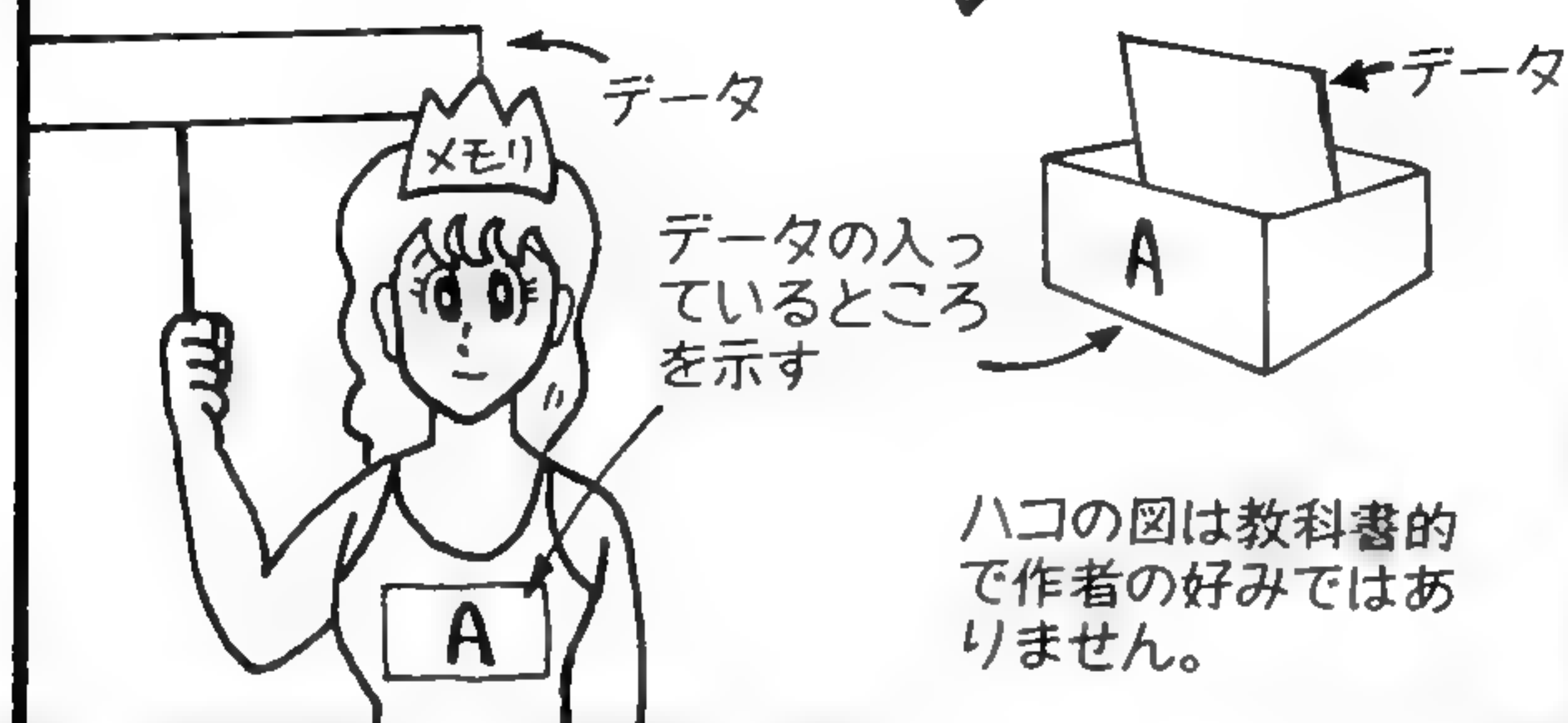



技術計算で、繰返し計算をする時は、精度面で注意する必要がありますね。

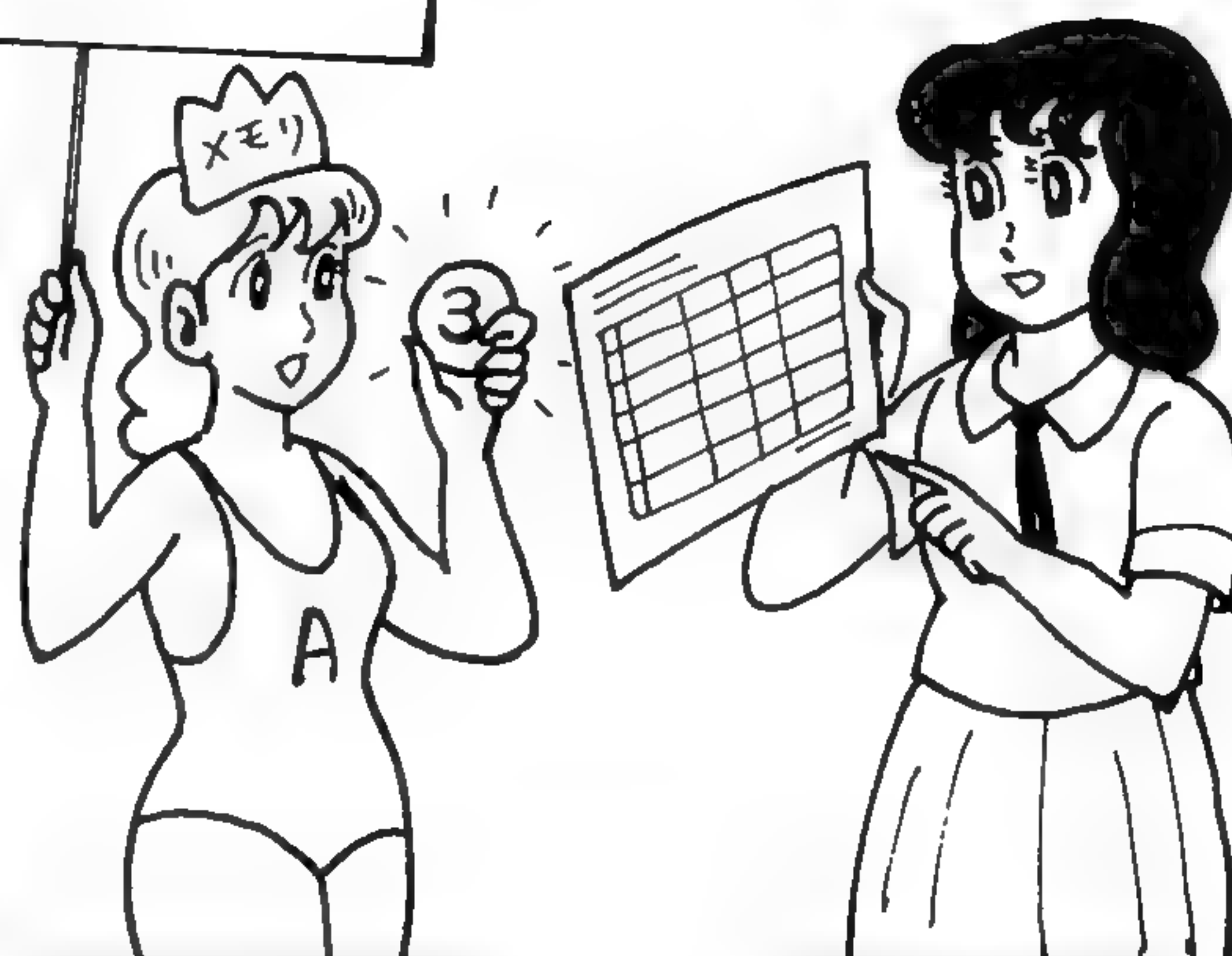





BASICにかぎらずマイコンがあつかうデータにはデータそのものと、そのデータを入れるところがあります。BASICではこのデータを入れておくところを文字で表わせました。



## ⑤配列がわかれば メモリもわかる

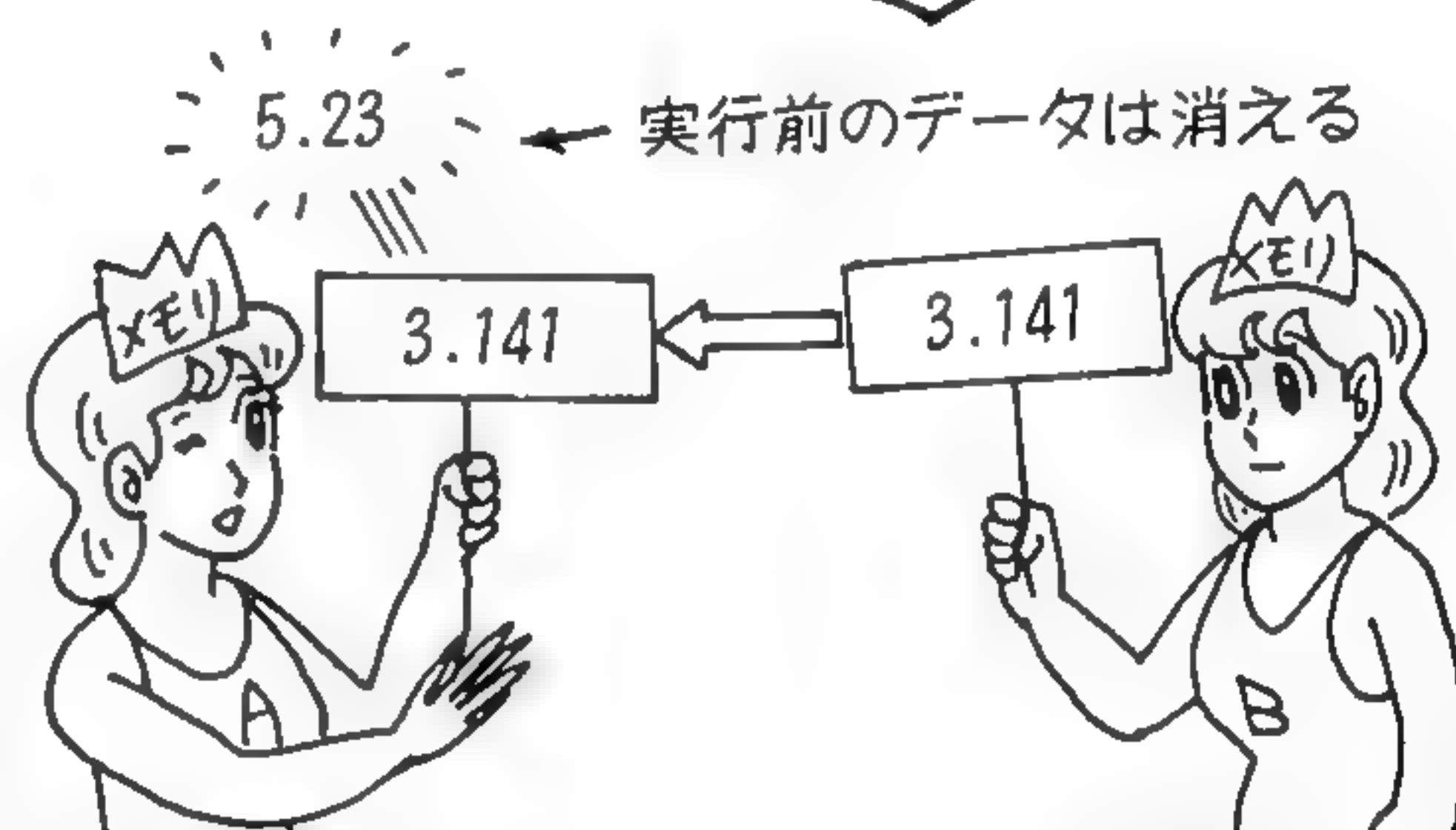


このAやBは制限はあるものの自由につけてよくABC=DEFでもいいのです。このA、B、ABC、DEFといったものはプログラム実行時中身がコロコロ変わるわけの変数と呼ばれます。式の中に出て来てプログラムで変化しないものは定数と呼ばれてます。

$$A = B + ABC + DEF + 2.217$$

変数                      定数

たとえばプログラムでA=Bとすれば、プログラムをRUNさせてこの文が実行されると、Bの入れものに入っているデータと同じものをAの入れものに入れるという動作をしますね。



一方先生は生徒別の点数表が必要であり、鈴木君の国語は何点、数学は何点……佐藤君の国語は何点、数学は何点……山田さんの国語は何点、数学は何点……田中さんの国語は何点、数学は何点……というような内容を表にします。これは二次元のデータになります。

氏名 科目	鈴木	佐藤	山田	田中
国語				
数学				
地理				
物理				



ところで私たちが日常取り扱うデータはたとえば生徒がもらう成績表では国語が何点、数学が何点といったものがありますが、これは一次元のデータと言われます。

科目	点数
国語	
数学	
地理	
物理	



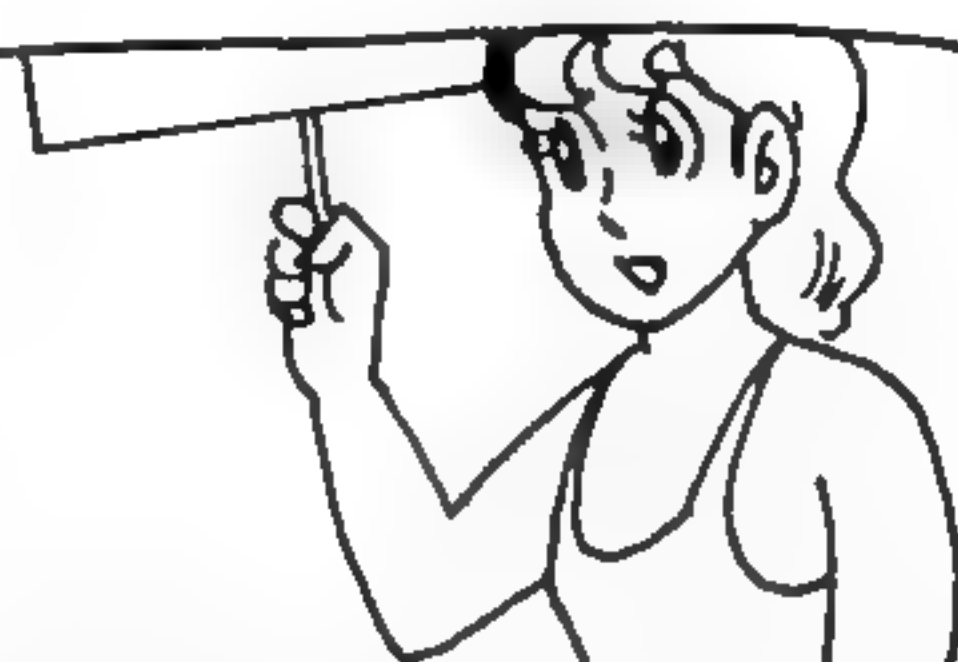


この変数の特徴はプログラムで使用する前にDIM(ディメンジョン)文で大きさを宣言しておく必要があることです。



一次元データ、二次元データなど耳なれない言葉と思いますが、ひんぱんに使われるコンピュータ用語ですからできれば覚えてください。この種のデータを扱う時に使って便利なのが配列変数なのです。

配列変数?



また DIM A(3, 2)にすると、12の変数が使えるようになります。

サイズ  
DIM A(3, 2)

...

二次元

A(0, 0) A(0, 1) A(0, 2)  
A(1, 0) A(1, 1) A(1, 2)  
A(2, 0) A(2, 1) A(2, 2)  
A(3, 0) A(3, 1) A(3, 2)

DIM A(11)で得られる12の変数とは用途が違います。

0から3までの限定変数

0から2までの限定変数

A(I, J)

A(0, 0)からA(3, 2)まで  
12種の変数



たとえば DIM A(4)と宣言すると、そのあとのプログラムで同じ名前で番号の違う5つの変数を使えるようになります。なぜ指定した4よりもひとつ多いのかというと0(ゼロ)番目もあるからです。

...

DIM A(4)

一次元

このあと  
A(0)  
A(1)  
A(2)  
A(3)  
A(4)  
の変数名が  
使える。

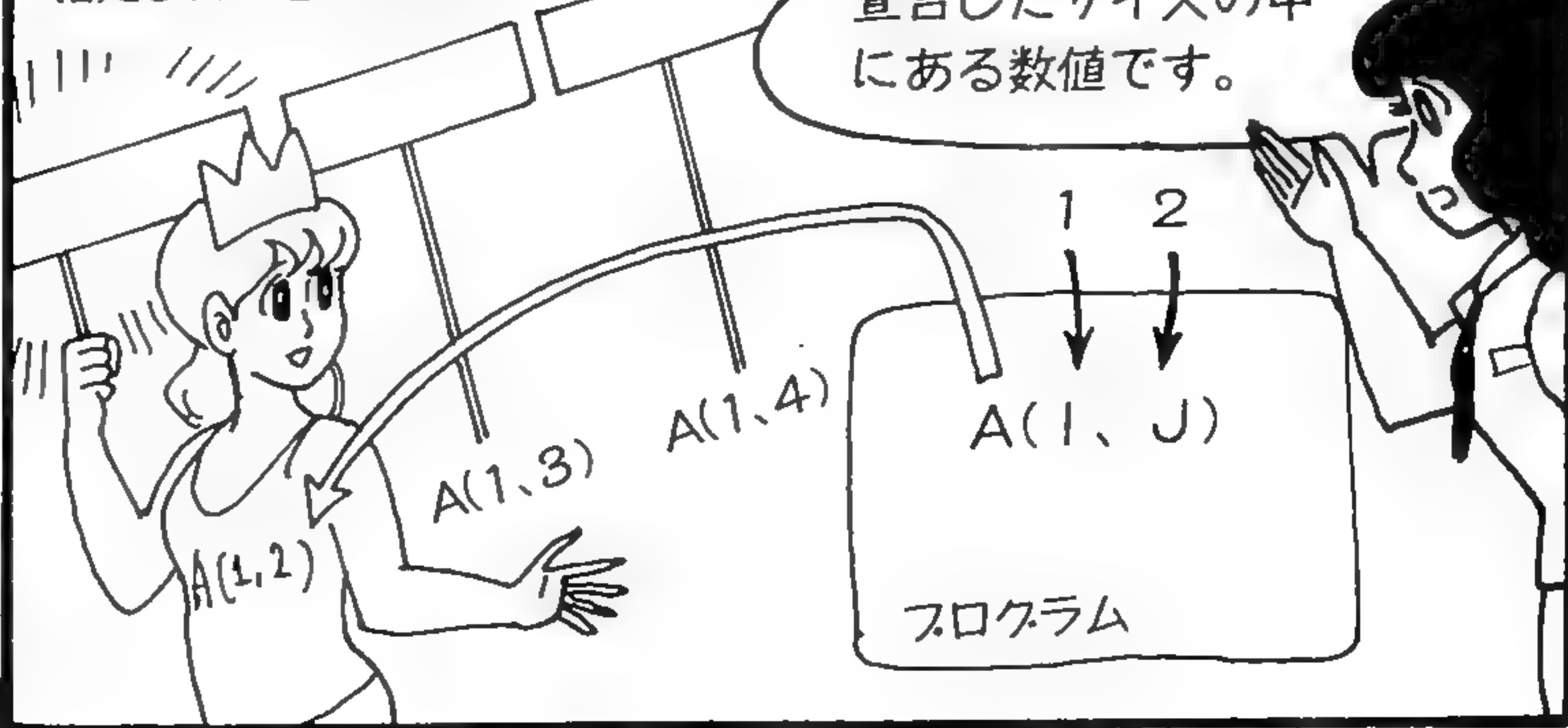


またDIM(3)は I = 3 : DIM(I)としても同じです。なんでこんなめんどろなことを思われるかも知れませんがFOR~NEXT文と併用すればスゴイことができるのです。



このデータを指定したことになる。

IとかJとかの中身は宣言したサイズの中にある数値です。





その指定は、変数名の次の( )の中で行ないます。

A( )

↑  
ここで何番目のメモリかを指定します。

それぞれのメモリは、変数Aの何番目という形で指定できるようになります。

A  
0 1 2 3

↑ 0番もある!

くりかえしますと、DIM A(3)を実行したあとは、Aという変数名で4つのメモリが与えられ、

A

↑ RAMメモリ

この場合Aのメモリは0番から始まって3番目なので、4以上の数字を入れるとエラーとなります。

DIM A(3)

エラーになります

A(4) = A(2) + 1

A(3)とすると、4番目のメモリが指定されます。

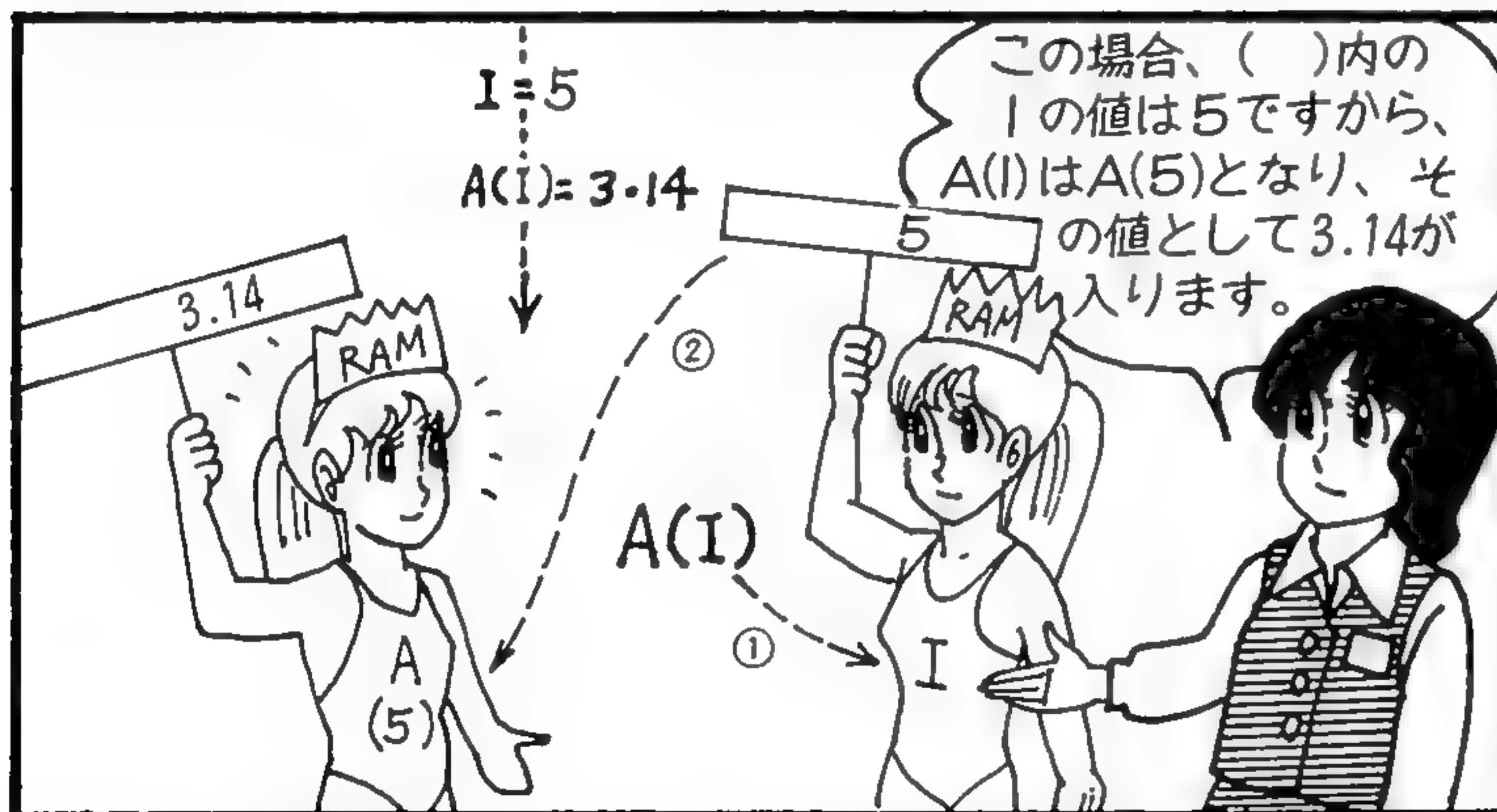
A  
0 1 2 3

↑  
A(3)

たとえば、A(1)にすると2番目のメモリが、

A  
0 1 2 3

↑  
A(1)



この( )の中は変数名を使っても、また式でもいいんです。

A(変数名)

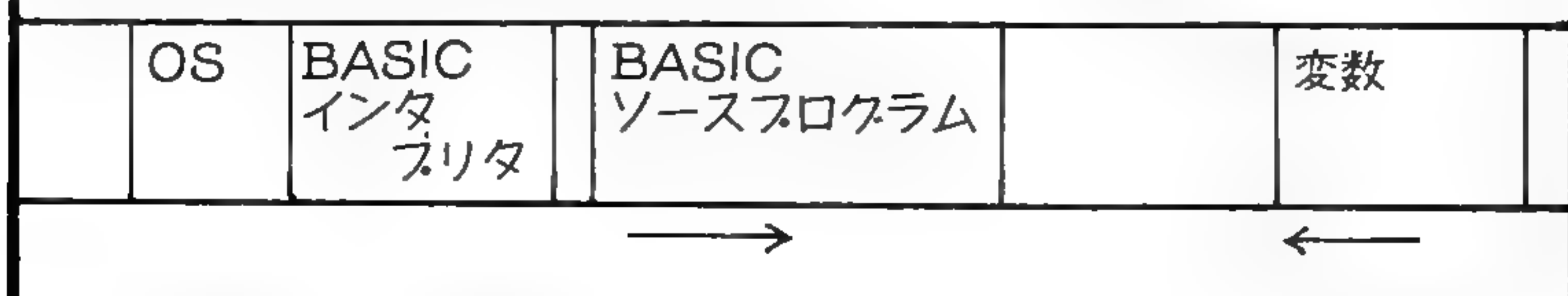
A(式)

パソコンのメインメモリは、システムを動かすOSプログラム、BASICインタプリタ、私たちがつくるBASICソースプログラム、そして変数域から成り立っています。配列も変数域に割り当てられます。

メインメモリ

←ROMまたはRAM→

←RAM→



( )内が式の場合は、まず式を計算して答えを出し、その答えで何番かのメモリを指定することになります。

A(I+A(1))

↑  
まずここが計算される



プログラム例です。

```

10 DIM A(3)
30 WAIT 50
50 FOR I=0 TO 3
60 PRINT "I="; USING "##"; I
70 INPUT " A(I)= "; A(I)
80 NEXT I
    
```

たとえば、インプットの時は次のようにできます。

配列はデータをひとまとめにしてメモリすることによってFOR~NEXT文でうまく処理できるようになっています。

```

FOR I=1 TO N
X=X+A(I)
NEXT I
    
```

PRINT "I="; USING "##"; I  
これは次の何番目のデータをインプットするのか確認できるようにするためのものです。

I= 0

##

WAIT 50 は表示する時間を指定するものです。ポケコンPC-1500 の場合は、これが必要です。

時間を限定しないと、ここでプログラムが止まってしまう。

```
PRINT "I="; I
```

プログラムを説明します。  
DIM A(3)は配列の宣言文です。メモリに割り当てた内容を書いておきましょう。

正義君のテストの成績

A		0	1	2	3
国語	数学	地理	物理		
83	95	88	98		

スゴイ!

PC-1500 RAMメモリ

A			
0	1	2	3
83	95	88	98

これでひとまずデータをポケコンの中に入れることができます。

0	1	2	3
国語	数学	地理	物理
83	95	88	98

INPUT "A(1)="; A(1)  
この文によってI=0からI=3までA(0)、A(1)、A(3)にインプットしたデータが入ります。

A(I)= 65

キー操作 [6] [5] [ENTER]

```

100 X=0
110 FOR I=0 TO 3
120 X=X+A(I)
130 NEXT I
140 X=X/4
    
```

やはりFOR~NEXT文を使います。この方法もコンピュータで平均値を求める時のひとつの定石です。

(スラッシュ) 割算の記号です。

このデータを使って、正義君の平均点を求めてみましょう。

$$X = \frac{A(0)+A(1)+A(2)+A(3)}{4}$$

これでもいいんですが、配列を使う意味が半減します。



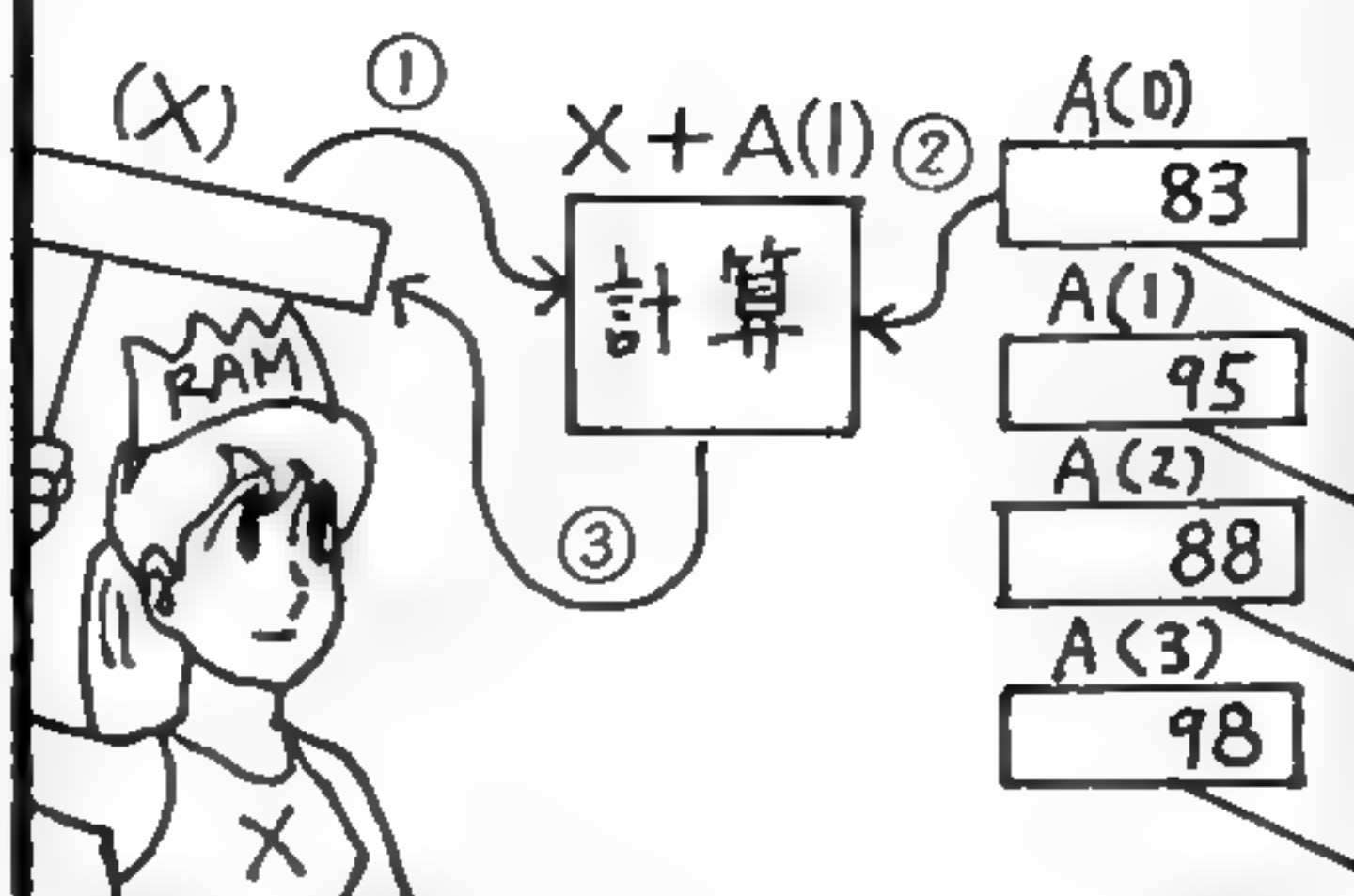
プログラムをRUNすると、FOR~NEXTには含まれた $X = X + A(I)$ は、次のように変化します。

$I = 0$ で

$$X = \begin{array}{c} X + A(0) \\ \uparrow \quad \downarrow \\ 83 \quad 83 \end{array}$$



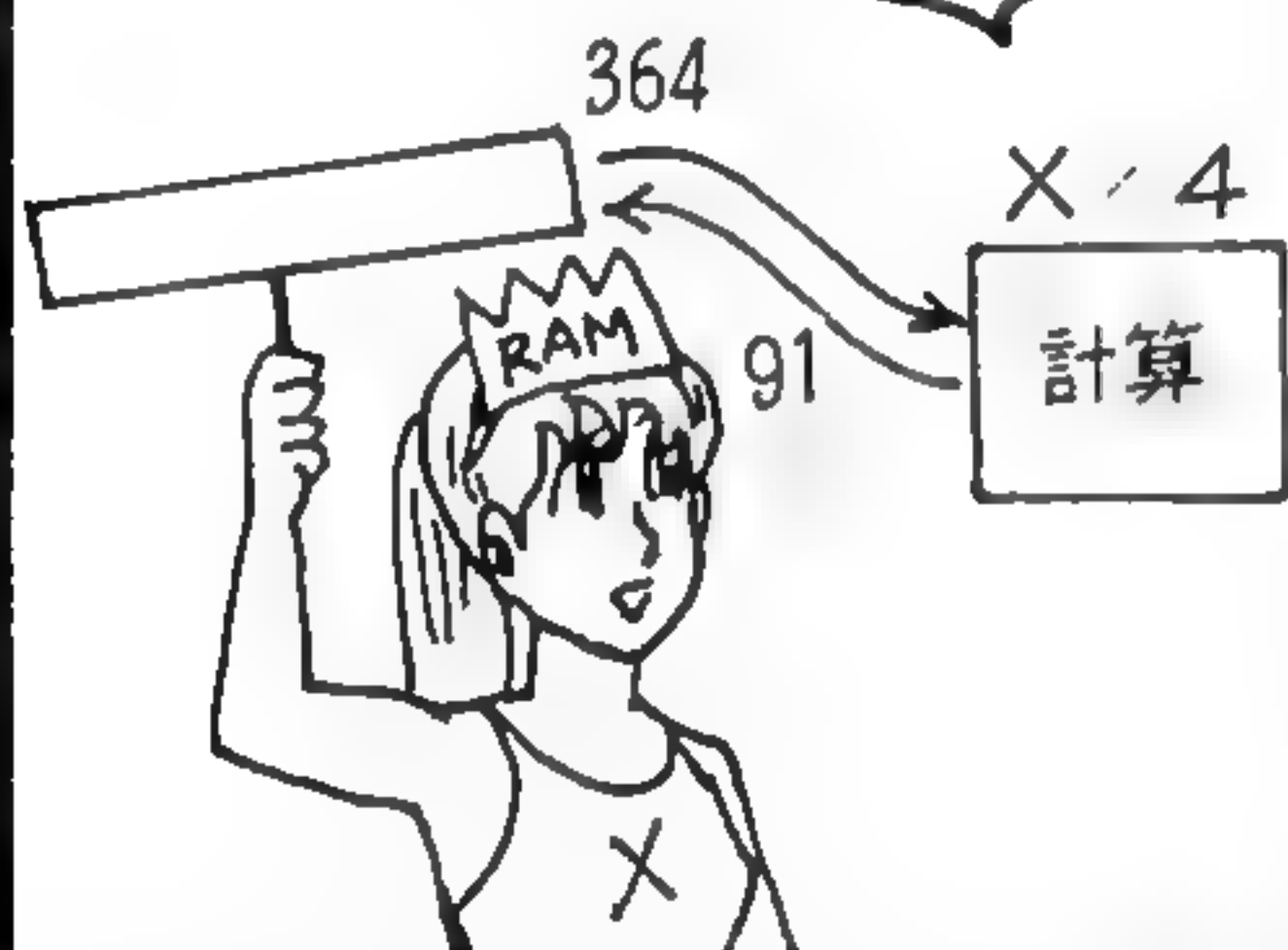
次の $X = X + A(I)$   
これが累加するための文です。



プログラムを説明します。 $X = 0$ 、これはFOR~NEXTの中で $X$ に累計値を入れていくので、最初にクリアして $X$ をゼロにしておく必要があります。

$X = 0$   
FOR  
NEXT

この $X$ を4で割ると平均値が求められます。  
つまり $X = X / 4$ です。



$I = 2$ で

$$X = \begin{array}{c} X + A(2) \\ \uparrow \quad \downarrow \\ 266 \quad 178 \quad 88 \end{array}$$

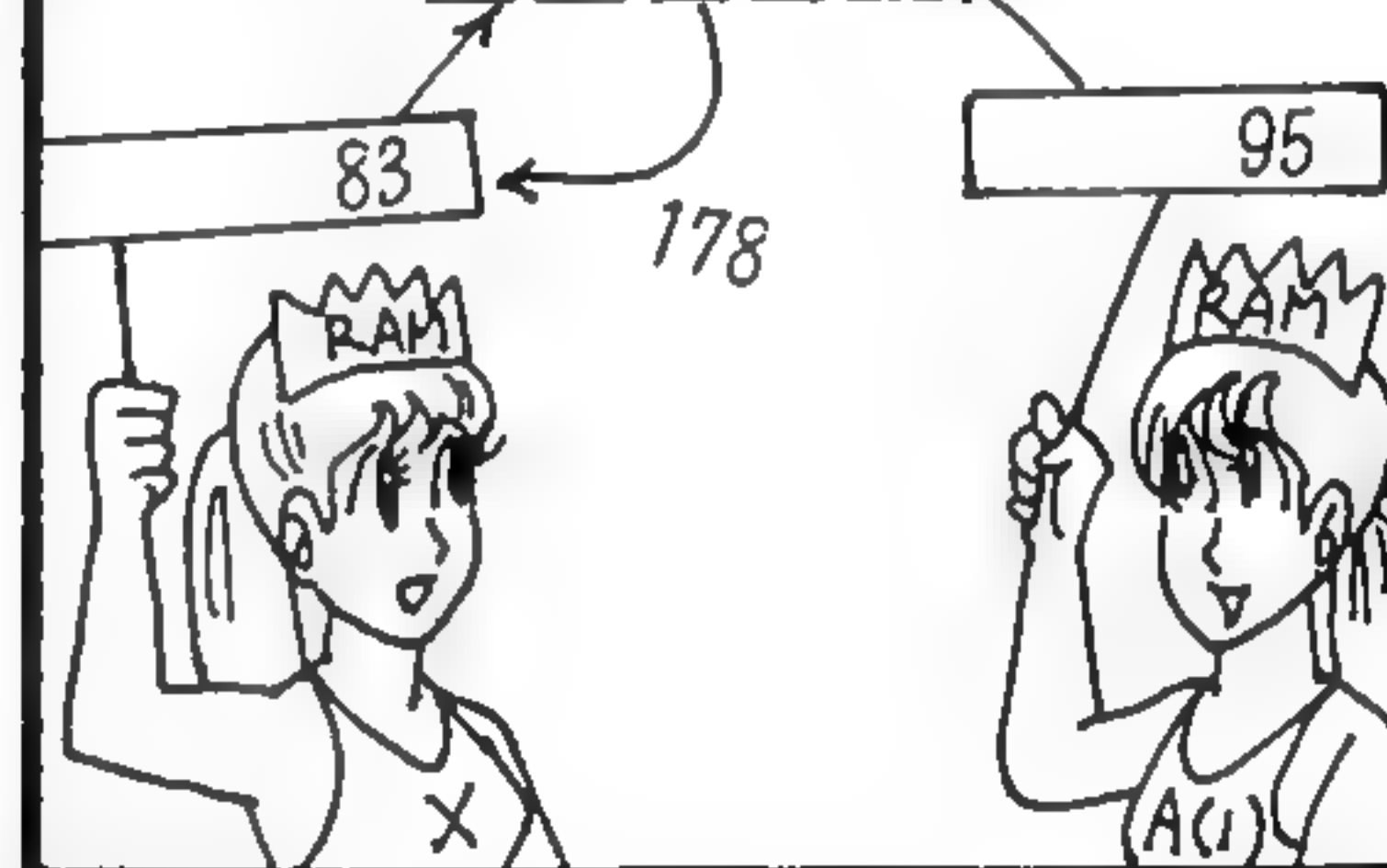
$I = 3$ で

$$X = \begin{array}{c} X + A(3) \\ \uparrow \quad \downarrow \\ 364 \quad 266 \quad 98 \end{array}$$

となり、FOR~NEXTを抜ける時には $X$ には364が入っています。

$I = 1$ で

$$X = \begin{array}{c} X + A(1) \\ \uparrow \quad \downarrow \\ 178 \quad 83 \quad 95 \end{array}$$



何度も出てきましたが、プログラムの式の左辺は、ひとつのメモリを示していて右辺の計算の結果をそのメモリに移せということなのです。移し終わると、左辺の内容は変わっています。

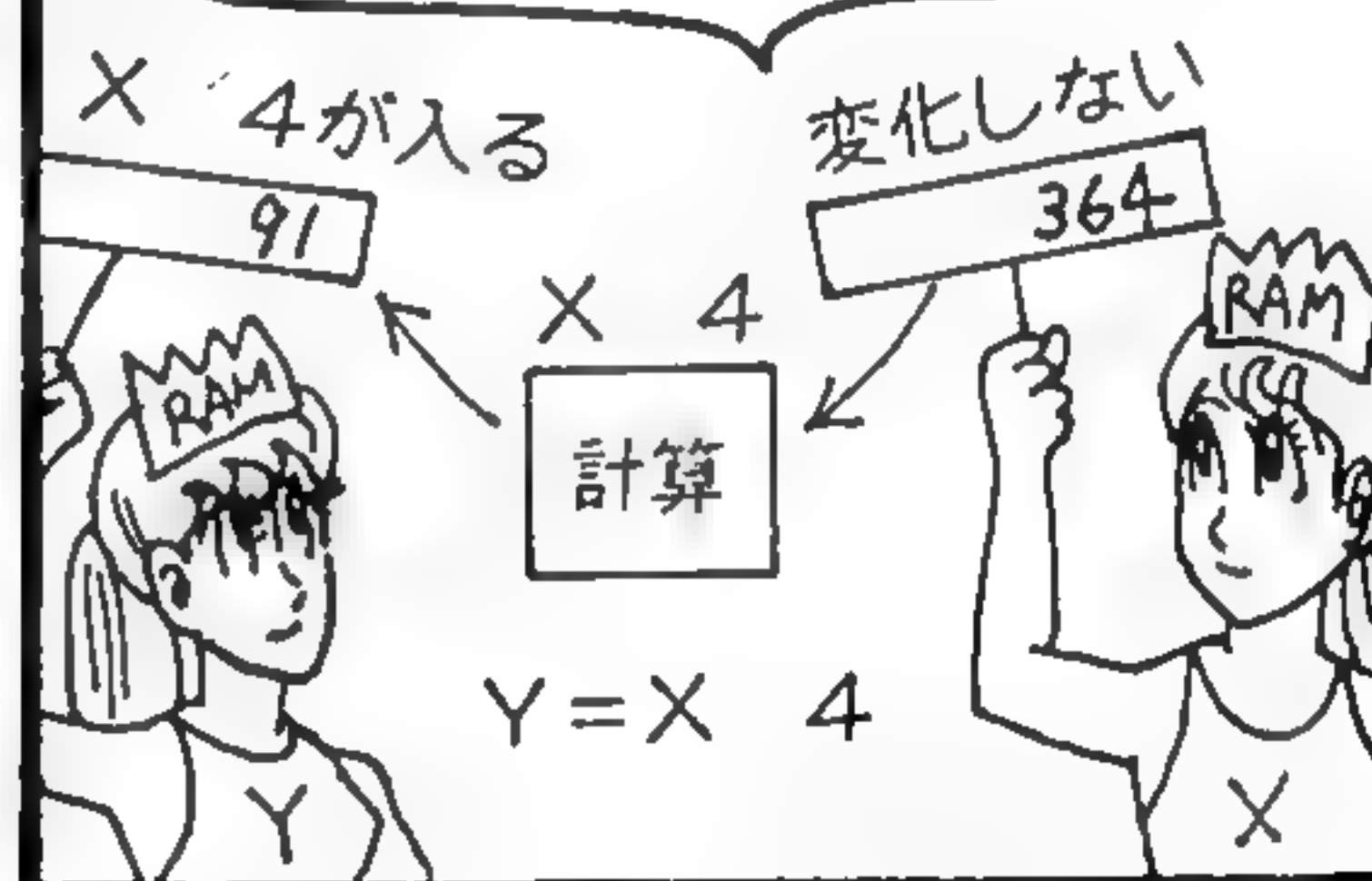
$$X = X + A(I)$$

たとえば、 $Y = X / 4$ とすると、 $Y$ に平均値が入り、 $X$ は元のままで変化しません。

$X$ : 合計  
 $Y$ : 平均値



ただ、合計も印字したいような場合は別の変数名を使って $X$ の内容をくずさないようにします。

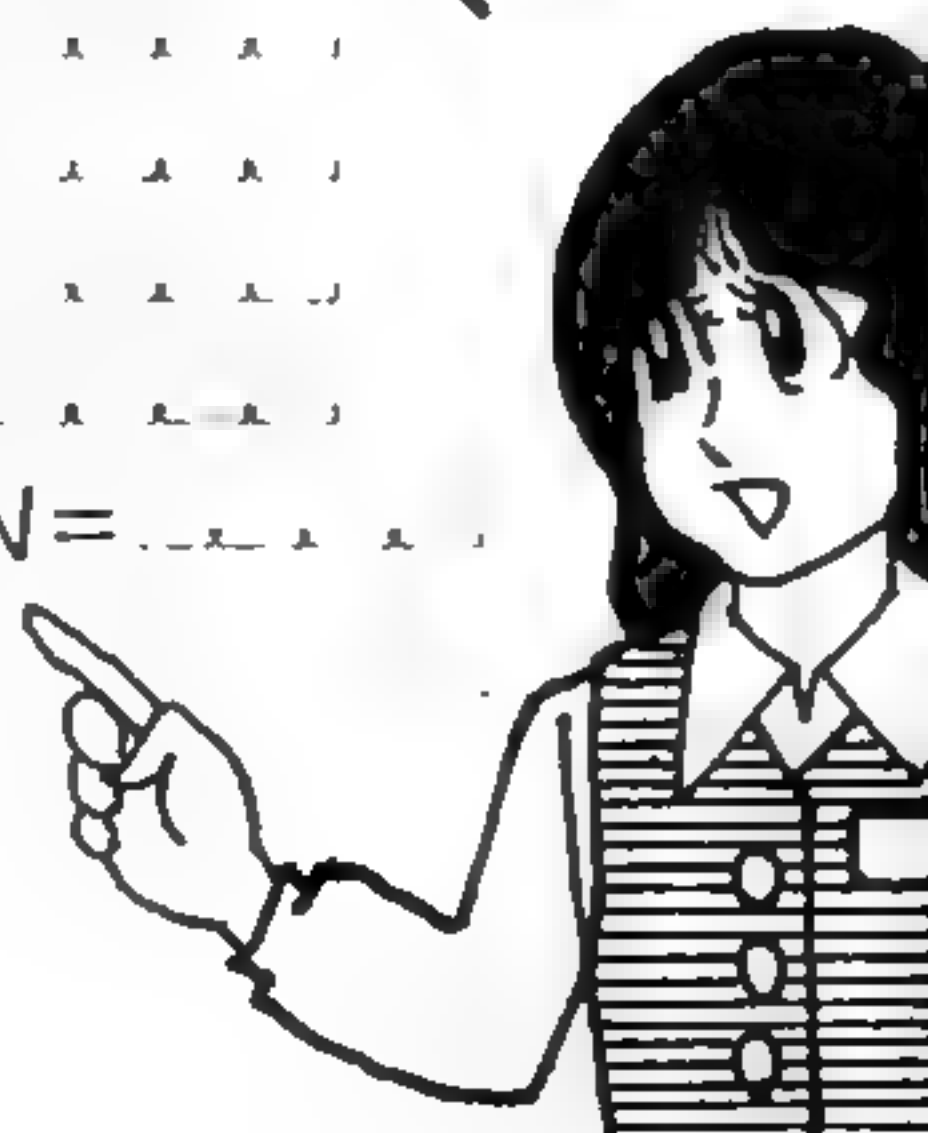


プログラム例です。

```
200 FOR I=0 TO 3
210 LPRINT "A("; USING
    "##"; I; ")=";
    USING "####"; A(I)
220 NEXT I
230 LPRINT "HEIKIN="; X
```

あとは、これをフォーマットを決めてプリントすればいいですね。

A(0)=  
A(1)=  
A(2)=  
A(3)=  
HEIKIN=



ですから、左辺に計算式があるというのは間違いです。左辺は右辺の計算結果を入れるところです。

$X \leftarrow X + A(I)$   
↑            ↑  
計算後      計算前





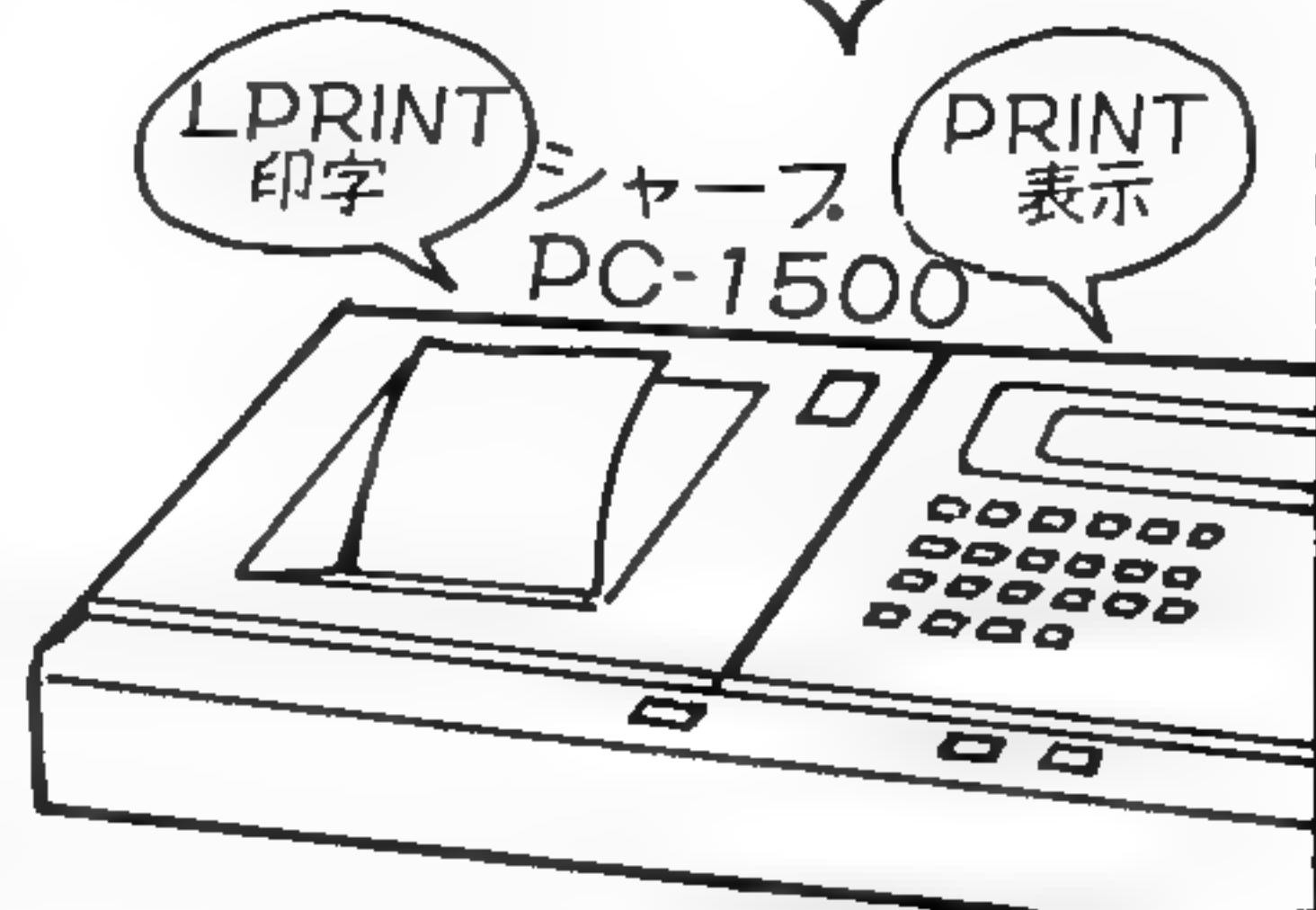
次のUSING '####';  
A(I)で、得点をプリントしま  
す。これがI=0から3まで  
4回繰り返されます。

A(0)= 83  
A(1)= 95  
A(2)= 88  
A(3)= 98  
↑    ↑  
符号   符号

その次にフォーマットが続き  
'A('USING '##';I;')='  
で A(I)= とプリント  
しようとしているので  
す。

"A("USING "##";I;")=";  
↓  
A( )=  
↑  
この1桁は正負の符号に使われる

プリントプログラムの説明を  
します。LPRINTはPC-15  
00のプリンタに出力せよと  
いう命令です。



このプログラムで、正義、悦  
子、拓人の3人の平均点を求  
めました。



TOKUTEN  
NAME=MASAYOSI  
A(0)= 83  
A(1)= 95  
A(2)= 88  
A(3)= 90  
HEIKIN= 89

TOKUTEN  
NAME=ETUKO  
A(0)= 85  
A(1)= 98  
A(2)= 89  
A(3)= 92  
HEIKIN= 91

TOKUTEN  
NAME=TAKUTO  
A(0)= 88  
A(1)= 99  
A(2)= 100  
A(3)= 96  
HEIKIN= 95

STATUS 1 253

```
10: DIM A(3)
20: INPUT "NAME=";
   A$
30: WAIT 50
50: FOR I=0 TO 3
60: PRINT "I=";
   USING "##"; I;
   BEEP 1
70: INPUT "A(I)=";
   A(I)
80: NEXT I
100: X=0
110: FOR I=0 TO 3
120: X=X+A(I)
130: NEXT I
140: X=X/4
180: LPRINT "TOKUTE
   N"
190: LPRINT "NAME="
   ;A$
200: FOR I=0 TO 3
210: LPRINT "A(";
   USING "##"; I;
   )="; USING "###
   #"; A(I)
220: NEXT I
230: LPRINT "HEIKIN
   =" ; X
240: END
```

それではこのプログラム  
をPC-1500 でインプ  
ットしてみます。



これがプログラムリストです。  
253バイトあります。NAMEを  
A\$として誰の得点かわかる  
ようにしまし  
た。BE  
EPは音を  
出す命令で  
す。



しかし、配列として最も  
多く使われるのは次にお話  
する2次元の配列でしょう。

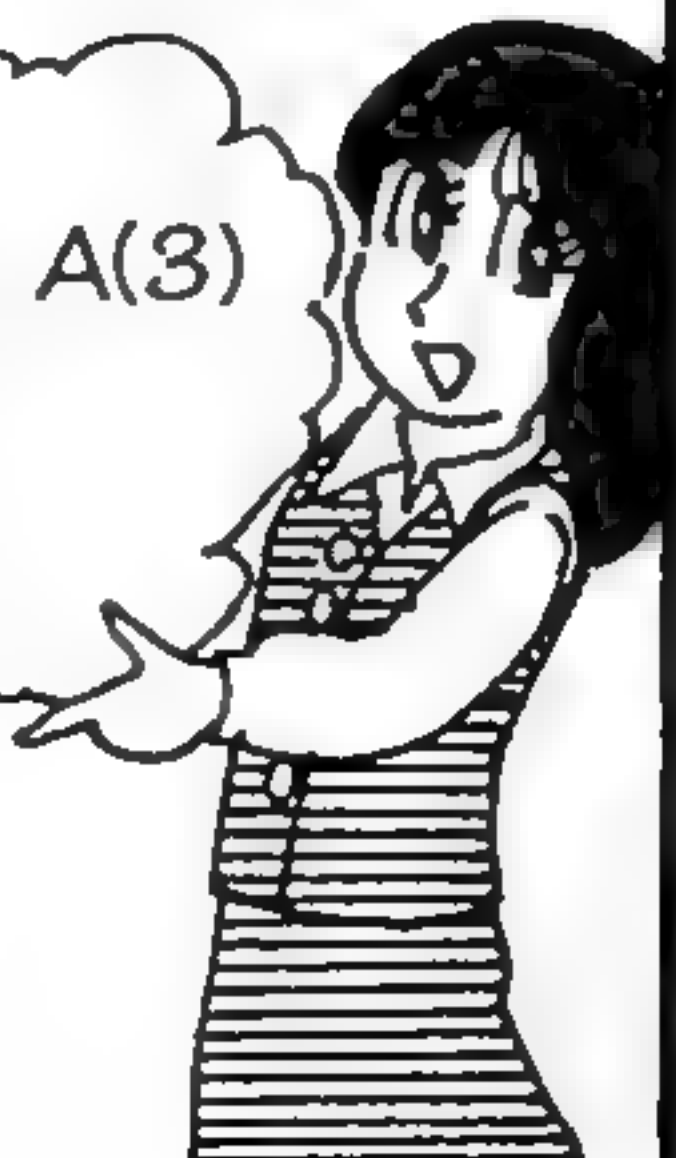
A(I)・・・一次元配列

A(I, J)・・・二次元配列



ここまでは1次元の配列につ  
いてのお話でした。

DIMENSION A(3)  
A(0), A(1),  
A(2), A(3)



ひとまとまりのデータを扱う  
時は、配列とFOR~NEXT  
を使って処理するのが一番で  
す。

X=X1+X2+X3.....  
単独の変数では  
FOR~NEXT  
は使えない









この番号10の文を実行した時、変数AXには、 $(3+1) \times (4+1) = 20$ のメモリが確保されます。

AX  
(0,0) (0,1) (0,2) (0,3) ...

まず、必要なメモリを確保するために配列の大きさを宣言します。変数名はAXとしましょう。

10 DIM AX(3,4)  
↑      ↑  
変数名 配列の  
         サイズ

そして変化させる手段がFOR~NEXT構文というわけです。

↓  
FOR I=1 TO N  
X=X+AX(I,J)  
↑  
NEXT I

そこでまずJを0から3まで変化させます。

↓  
FOR J=0 TO 3  
PRINT "I="; USING "##"; I; "J="; J; BEEP 1  
INPUT "AX(I,J)="; AX(I,J)  
↑  
NEXT J

インプットの順として、個人別の科目の得点を入れることにすると表を縦に入れていくことになります。

次に表のデータをこのAXの中にインプットする必要があります。


30 WAIT 50  
50 FOR I=0 TO 2  
60 FOR J=0 TO 3  
70 PRINT "I="; USING "##"; I; "J="; J; BEEP 1  
80 INPUT "AX(I,J)="; AX(I,J)  
90 NEXT J  
100 NEXT I

これもよく使うパターンですよ。

まとめて書くところになります。

次にIを0から2まで変化させればよいのです。

↓  
FOR I=0 TO 2  
AX(I,J)  
↑  
NEXT I

AX(I,J)		I → 0 1 2 3			
J ↓					平均A
0					0
1					0
2					0
3					0
4	平均B	0	0	0	0

次に平均値を求めるわけですが、まず累計しなくてはならないので、使うメモリを0にしておく必要があります。

AX(3,0)=0  
AX(3,1)=0  
AX(3,2)=0  
⋮

インプットの順を間違えないようにしなければいけません。




次に横方向にクリアします。

0				
1				
2				
3				
4		0	0	0

1 → 0 1 2 3

```
FOR I=0 TO 2
  AX(I,4)=0
NEXT I
```

クリアする時もFOR~NEXT文を使うとスマートになります。まず縦方向にクリアします。

			0
			0
			0
			0
			0

↓ J 0 1 2 3 4

```
FOR J=0 TO 4
  AX(3,J)=0
NEXT J
```

1 2 3

まず横に累計するために1を0から2まで変化させます。この間Jは固定しています。

	0	1	2	3
↓ J	---	---	---	累計

```
FOR I=0 TO 2
  AX(3,J)=AX(3,J)+AX(I,J)
NEXT I
```

↑ この1が横累計のカギです。

文番号をつけてできあがりです。この番号は適当につけています。

```
150 FOR J=0 TO 4
160 AX(3,J)=0
170 NEXT J
180 FOR I=0 TO 2
190 AX(I,4)=0
200 NEXT I
```

この場合は10とびです。

この部分が実行されるとメモリAX(3,J)には科目別累計が入っていることになります。

AX(3,0)	
AX(3,1)	
AX(3,2)	
AX(3,3)	

1回横累計が終わるたびに今度はJを変化させます。これで科目別平均が順々に求められます。

	0	1	2	3
↓ J	---	---	---	---
0				
1				
2				
3				
4				

```
FOR J=0 TO 3
  ↓
  AX(3,J)=AX(3,J)+AX(I,J)
NEXT J
```

```
300 FOR J=0 TO 3
310 FOR I=0 TO 2
320 AX(3,J)=AX(3,J)+AX(I,J)
330 NEXT I
340 NEXT J
350 FOR I=0 TO 2
360 FOR J=0 TO 3
370 AX(I,4)=AX(I,4)+AX(I,J)
380 NEXT J
390 NEXT I
```

↑ 横のカギ

↑ 縦のカギ

縦横まとめてプログラムしました。

縦の場合も、説明は省きますが、同じ考え方でプログラミングできます。

	0	1	2	3
↓ J	↓	↓	↓	↓
0				
1				
2				
3				
4				



	→ 1	3	J
2			
			0
			1
			2
			3
			4
		総平均	

その平均値を縦に累計して  
4で割り総平均を求めれば  
この計算は終了です。

```

550 FOR J=0 TO 3
560 AX(3,4)=AX(3,4)+AX(3,J)
570 NEXT J
580 AX(3,4)=AX(3,4)/4
                    
```

累計を縦・横それぞれの  
個数で割れば平均が求め  
られます。

```

450 FOR J=0 TO 3
460 AX(3,J)=AX(3,J)/3
470 NEXT J

480 FOR I=0 TO 2
490 AX(I,4)=AX(I,4)/4
500 NEXT I
                    
```

ポケコンPC-1500のフリ  
ンタの桁数は18桁です。

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18

これで必要な答えはすべて配  
列AXの中に入っています。  
あとは、これをプリントすべ  
ばよいのです。

LPRINT

得点表印字フォーマット

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
T	O	K	U	T	E	N											
								M				E					T
K	O	K	U	G	O	x	x	x	x	o	o	o	o	x	x	x	x
S	U	G	A	K	U	o	o	o	o	x	x	x	x	o	o	o	o
T	H	I	R	I		x	x	x	x	o	o	o	o	x	x	x	x
B	U	T	U	R	I	o	o	o	o	x	x	x	x	o	o	o	o
K	O	B	E	T	U												
M	A	S	A	Y	O	S	I	=	o	o	o	o					
E	T	U	K	O				=	x	x	x	x					
T	A	K	U	T	O			=	o	o	o	o					
K	A	M	O	K	U	B	E	T	U								
K	O	K	U	G	O	=	o	o	o	o							
S	U	G	A	K	U	=	x	x	x	x							
T	H	I	R	I		=	o	o	o	o							
B	U	T	U	R	I	=	x	x	x	x							
S	O	U	H	E	I	K	I	N	=	o	o	o	o				

プリントはプログラ  
ムの総決算です。

ただメモリの内容を  
打ち出しただけでは  
だめですね。

誰が見ても良  
くわかるよう  
に工夫する必  
要があります。

表中の  
○○○や×××  
はメモリ内容を  
プリントするこ  
ろです。

␣はスペー  
スです。桁  
合せのために  
入れることが  
あります。



```

10: DIM AX(3, 4)
30: WAIT 50
50: FOR I=0 TO 2
60: FOR J=0 TO 3
70: PRINT "I=";
    USING "##"; I; "
    J="; J; BEEP 1
80: INPUT "AX(1, J)
    =" ; AX(1, J)
90: NEXT J
100: NEXT I
150: FOR J=0 TO 4
160: AX(3, J)=0
170: NEXT J
180: FOR I=0 TO 2
190: AX(1, 4)=0
200: NEXT I
300: FOR J=0 TO 3
310: FOR I=0 TO 2
320: AX(3, J)=AX(3, J)
    +AX(1, J)
330: NEXT I
340: NEXT J
350: FOR I=0 TO 2
360: FOR J=0 TO 3
370: AX(1, 4)=AX(1, 4)
    +AX(1, J)
380: NEXT J
390: NEXT I
450: FOR J=0 TO 3
460: AX(3, J)=AX(3, J)
    /3
470: NEXT J
480: FOR I=0 TO 2
490: AX(1, 4)=AX(1, 4)
    /4
500: NEXT I
550: FOR J=0 TO 3
560: AX(3, 4)=AX(3, 4)
    +AX(3, J)
570: NEXT J
580: AX(3, 4)=AX(3, 4)
    /4
600: LPRINT "TOKUTE
    N"
610: LPRINT TAB 8; "
    M   E   T"
620: LPRINT "KOKUGO
    "; USING "####"
    ; AX(0, 0); AX(1,
    0); AX(2, 0)
630: LPRINT "SUGAKU
    "; AX(0, 1); AX(1
    , 1); AX(2, 1)
640: LPRINT "THIRI
    "; AX(0, 2); AX(1
    , 2); AX(2, 2)
650: LPRINT "BUTURI
    "; AX(0, 3); AX(1
    , 3); AX(2, 3)
660: LF 1
670: LPRINT "KOBETU
    HEIKIN"
680: LPRINT "MASAYO
    SI="; AX(0, 4)
690: LPRINT "ETUKO
    =" ; AX(1, 4)
700: LPRINT "TATUTO
    =" ; AX(2, 4)
710: LF 1
720: LPRINT "KAMOKU
    BETU HEIKIN"
730: LPRINT "KOKUGO
    =" ; AX(3, 0)
740: LPRINT "SUGAKU
    =" ; AX(3, 1)
750: LPRINT "THIRI
    =" ; AX(3, 2)
760: LPRINT "BUTURI
    =" ; AX(3, 3)
770: LF 1
780: LPRINT "SQUEIK
    IN=" ; AX(3, 4)
790: END

```

インプット  
ゼロリセット  
横累計  
縦累計  
横平均  
縦平均  
横平均  
累計  
総平均  
プリント

600 LPRINT "TOKUTEN"

610 LPRINT TAB 8; "M E T"

620 LPRINT "KOKUGO"; USING "####";

AX(0,0); AX(1,0); AX(2,0)

630 LPRINT "SUGAKU"; AX(0,1); AX(1,1); AX(2,1)

640 LPRINT "THIRI"; AX(0,2); AX(1,2); AX(2,2)

650 LPRINT "BUTURI"; AX(0,3); AX(1,3); AX(2,3)

660 LF 1

670 LPRINT "KOBETU HEIKIN"

680 LPRINT "MASAYOSI="; AX(0,4)

690 LPRINT "ETUKO="; AX(1,4)

700 LPRINT "TAKUTO="; AX(2,4)

710 LF 1

720 LPRINT "KAMOKUBETU HEIKIN"

730 LPRINT "KOKUGO="; AX(3,0)

740 LPRINT "SUGAKU="; AX(3,1)

750 LPRINT "THIRI="; AX(3,2)

760 LPRINT "BUTURI="; AX(3,3)

770 LF 1

780 LPRINT "SOUHEIKIN="; AX(3,4)

プリントプログラムの例です。

プリント幅が足りないの  
でこんなフ  
ォーマット  
になりました。



プリント結果です。

TOKUTEN

	M	E	T
KOKUGO	83	85	88
SUGAKU	95	98	99
THIRI	88	89	100
BUTURI	90	92	96

KOBETU HEIKIN

MASAYOSI= 89

ETUKO = 91

TATUTO = 95

KAMOKUBETU HEIKIN

KOKUGO= 85

SUGAKU= 97

THIRI = 92

BUTURI= 92

④  
SOUHEIKIN= 91

リストは次のコマを見てください。



LPRINT...  
LPRINT...  
LPRINT...

LPRINT...  
LPRINT...

何回も繰  
り返してイ  
ンプット  
する命令に  
は、簡単にリビ  
ートできるよ  
うに考えてあり  
ます。

PC-1500 ポケ  
コンではリザー  
ブ機能を使います。





生徒名 科目名 テスト回数  
A(I, J, K)

二次元の成績表にテストの回数の要素(K)を加えると、このように書くことができます。

	正義 I=0		悦子 I=1		拓人 I=2	
	前 K=0	後 K=1	前 K=0	後 K=1	前 K=0	後 K=1
国語 J=0						
数学 J=1						
地理 J=2						
物理 J=3						

一次元A(I)、二次元A(I, J)とききましたので、三次元のモデルを考えてみましょう。

A(I, J, K)

この調子でいくと何次元の配列でもOKということになりますが、問題もあります。

A(I, J, K...  
...X, Y, Z)

クラス  
A(I, J, K, L)

さらにクラス別(L)といった要素を入れると四次元のモデルができます。

L=0 L=1 L=2

Aクラス Bクラス Cクラス

配列で大きなサイズを宣言してしまうと、それだけプログラムのメモリできる領域が減ってしまうのです。

プログラム域 変数域

ERROR

もうひとつはメインメモリの容量です。プログラムがまずメモリされ、次にプログラム実行時に変数域が大きいアドレスの方から形成されていきます。

プログラム 変数

メモリ余裕 プログラム実行時

ひとつはインタプリタの能力です。たとえば、ポケコンPC-1500では、現在のところ配列のサイズは二次元まで、変数名は2桁までです。

DIM A(0, 3, 2)

ERROR

適当にひとまとめにして、メインメモリから外部メモリにデータを移します。カセットテープを使うことができます。

また、いたずらに配列のサイズを大きくすることは考えものです。

カセットテープ

l=0 l=1 l=2

K=0 K=1 K=2 K=3  
A(1, J) A(1, J) A(1, J) A(1, J)

レコード

メインメモリ

プログラム A(I, J)

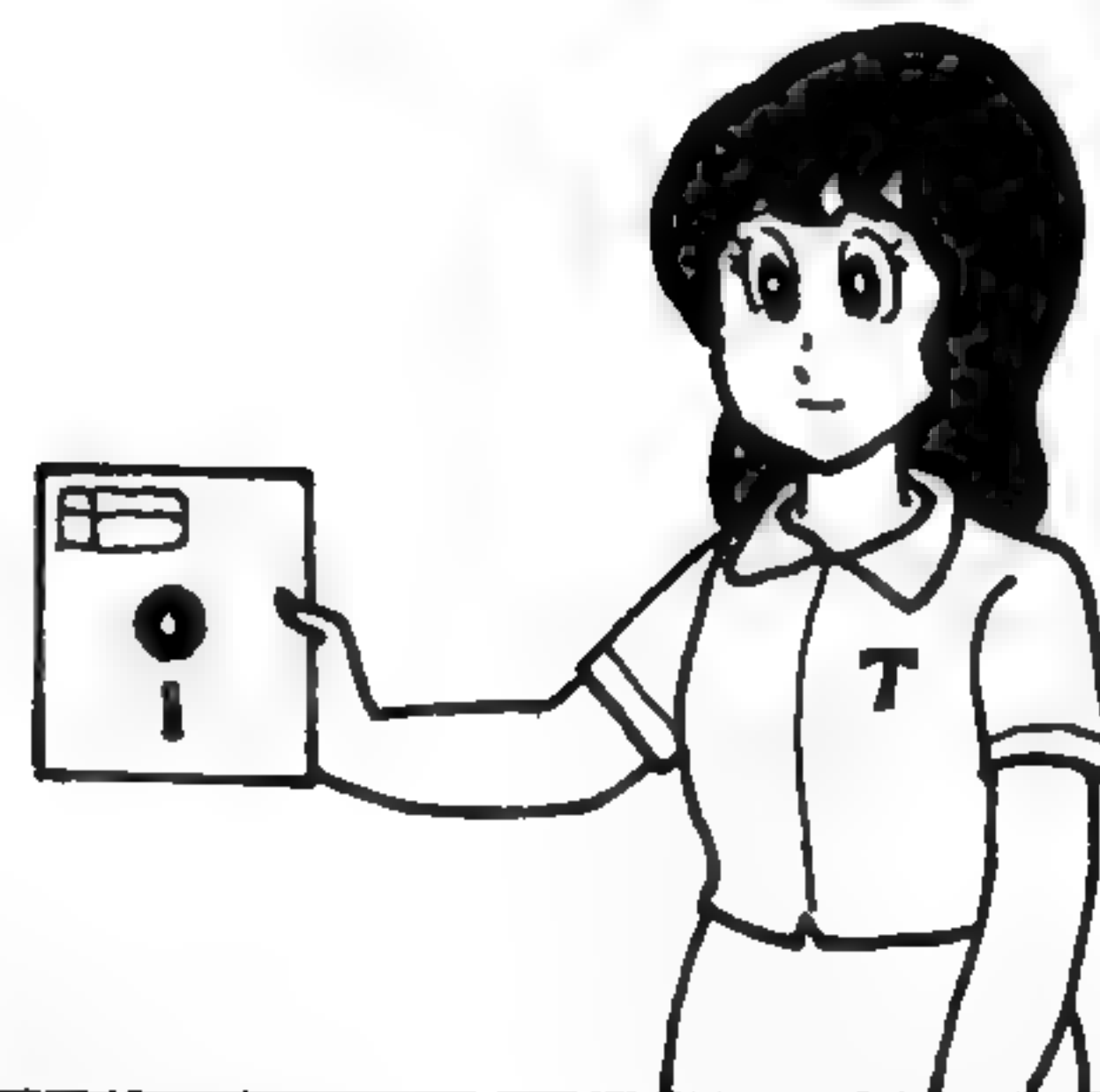
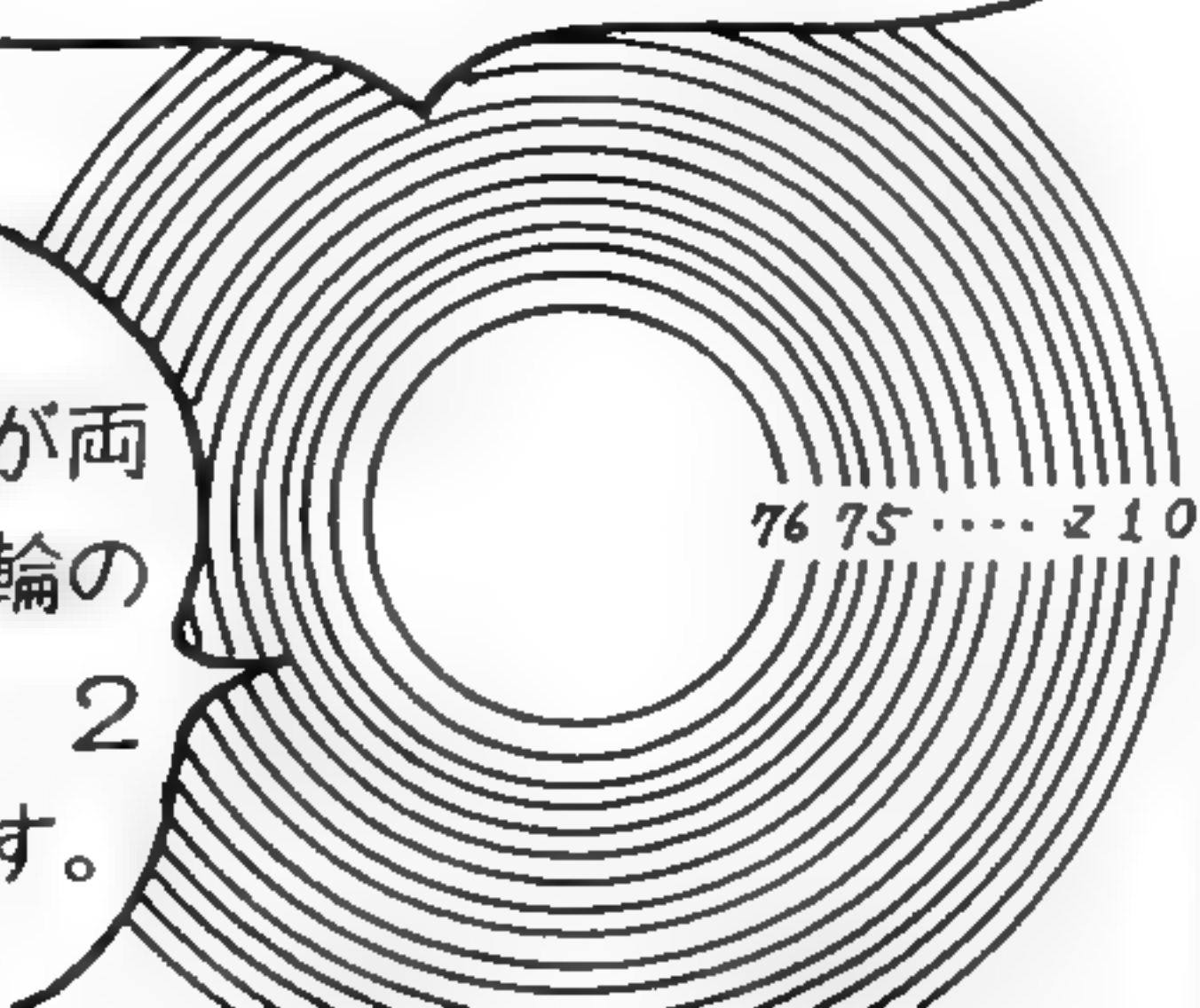
そこで配列のサイズ、プログラムの大きさを考え、調整する必要がある時もあります。



## ⑥フロッピーに データファイルをつくる

フロッピーディスク (FDと略す) には3インチ、3.5インチ、5インチ、8インチのがありますが8インチを例にとってお話ししましょう。

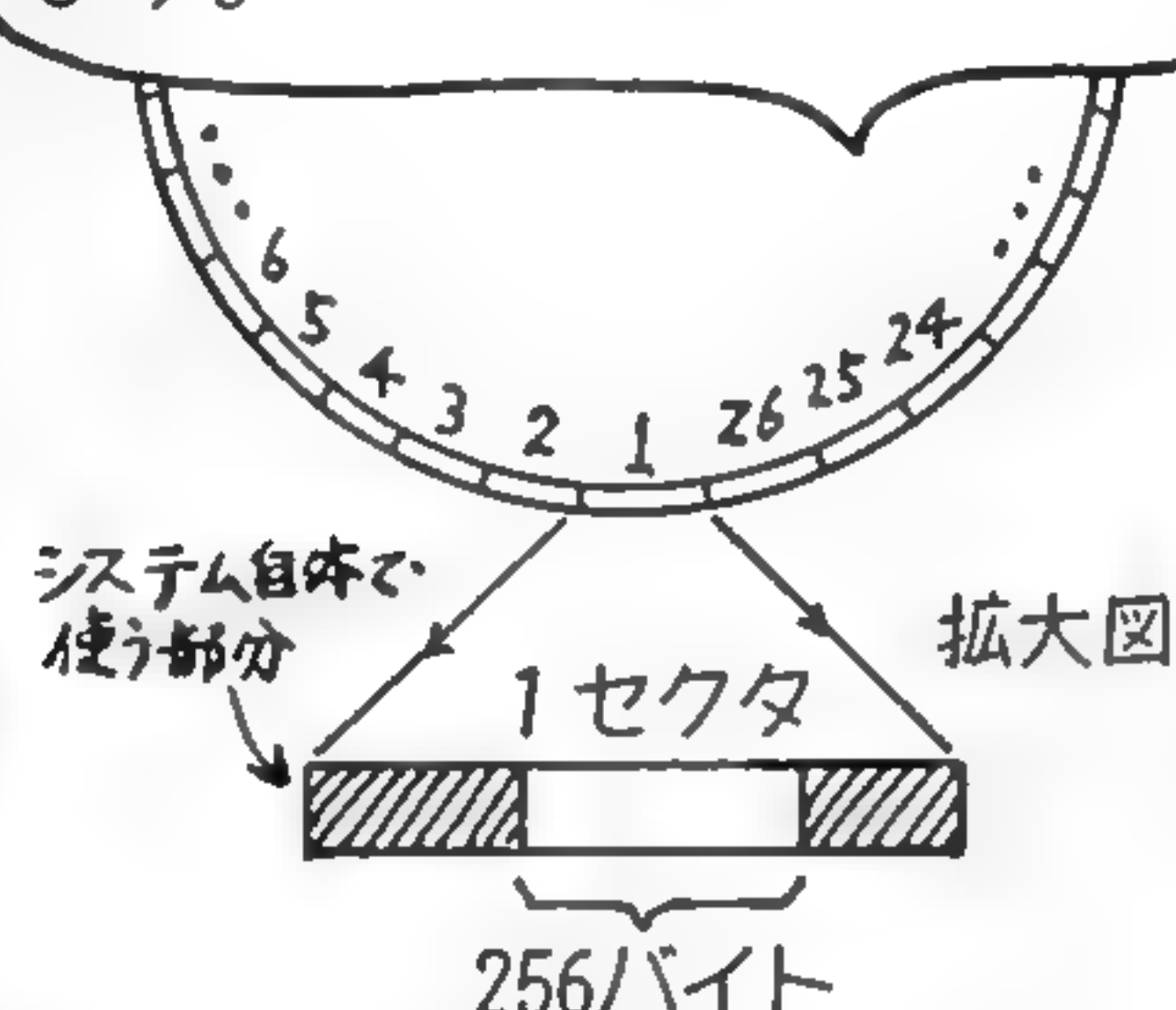
8インチフロッピーの中には77個の輪が両面についています。輪の番号は外から0、1、2……76となっています。



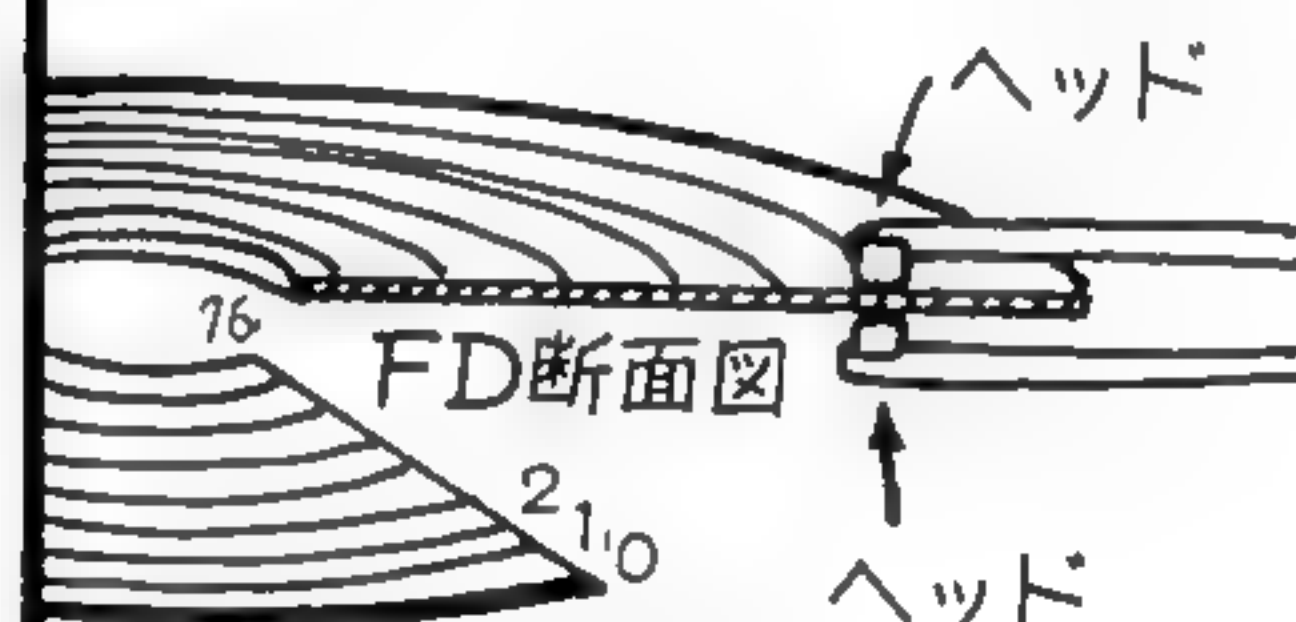
そこで8インチの場合は、これを計算すると1M(メガ)バイトの情報を記憶できるということになります。

256バイト/セクタ×  
26セクタ/トラック×  
2トラック/シリンダ×  
77シリンダ=  
1025024バイト=1Mバイト

このトラックはさらに26コのセクタに区切られており、1セクタは256バイト記憶できます。



この場合、77のシリンダがあると言い、1シリンダには2トラック(表と裏にひとつずつ)あります。ひとつの輪をトラックと言っています。



ランダムファイルは効率よく使えますがいろんな手続きがあってマニュアルを見ただけではむずかしいと思います。

あいている所に次のデータが書き込まれます。



ランダムファイル

フロッピーでつくるデータファイルにはシーケンシャルファイルとランダムファイルがあります。

順々に書き込まれます。 次のデータはここに書き込まれます。



シーケンシャルファイル

今まではプログラムのセーブやロードに使っていたフロッピーで、データのファイルをつくってみませんか。

データファイルはどう使うか？

プログラムファイルはSAVEとLOADで使える

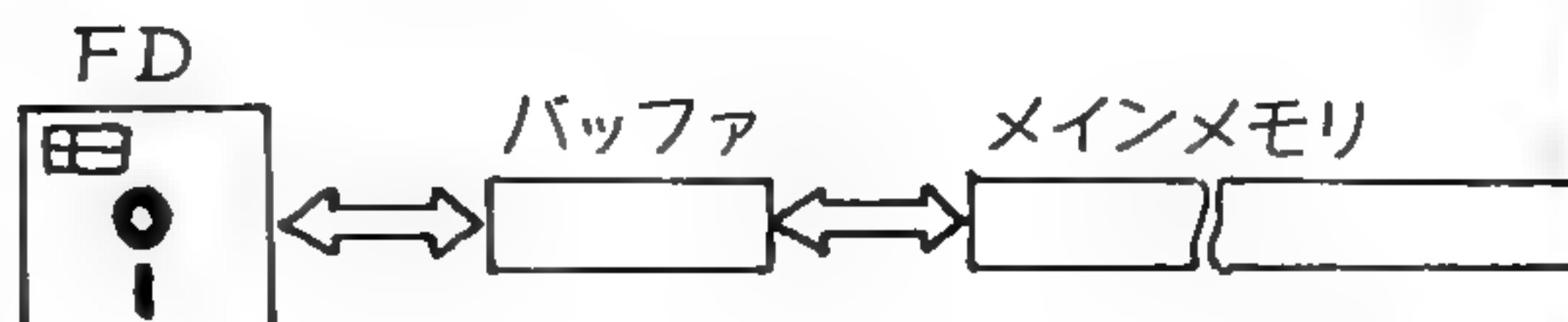


おもしろそう。



ここでマニュアルではもうひとつわからなかったことが明らかになるでしょう。さらに文字列の結合や分断もプログラムのテクニックにして使われています。程度は相当高いと思います。

また話の中に出てくるバッファというのは2つの速度の違うメモリの間においてデータの受渡しの調整をするところです。バッファレジスタとか言うことがあります。





まず、OPEN文を使ってディスク・ファイルをオープンします。この文が実行されるとディスク使用状態になります。

OPEN "2:ETUKO" AS #1

そして、この文の意味は、



この説明のステートメントやファンクションはOKI-BASICのもので、他のパソコンでも考え方は同じです。



作成するファイル名をここではETUKOとすることになります。

1セクタ  
1レコード  
256バイト

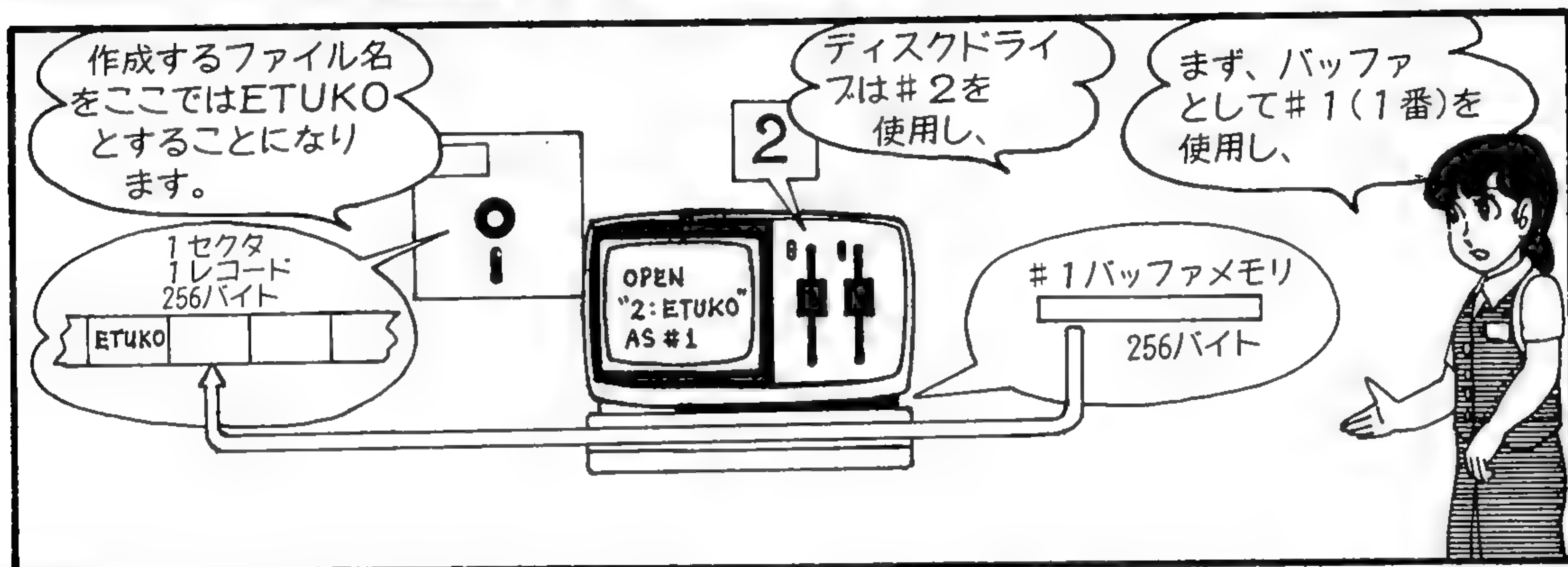
2

ディスクドライブは#2を使用し、

まず、バッファとして#1(1番)を使用し、

#1バッファメモリ  
256バイト

OPEN "2:ETUKO" AS #1



ではFIELD(フィールド)文から説明します。これはファイル中のレコードのフォーマットを決めるものです。1レコードは256キロバイト、バッファと同じ長さです。

FIELD #n, a AS x1,  
b AS x2, ...



ランダムファイル活用のためのプログラム文です。

RSET文  
MK\$  
PUT文  
GET文  
CVS  
RIGHT\$  
LEFT\$  
MID\$  
FIELD

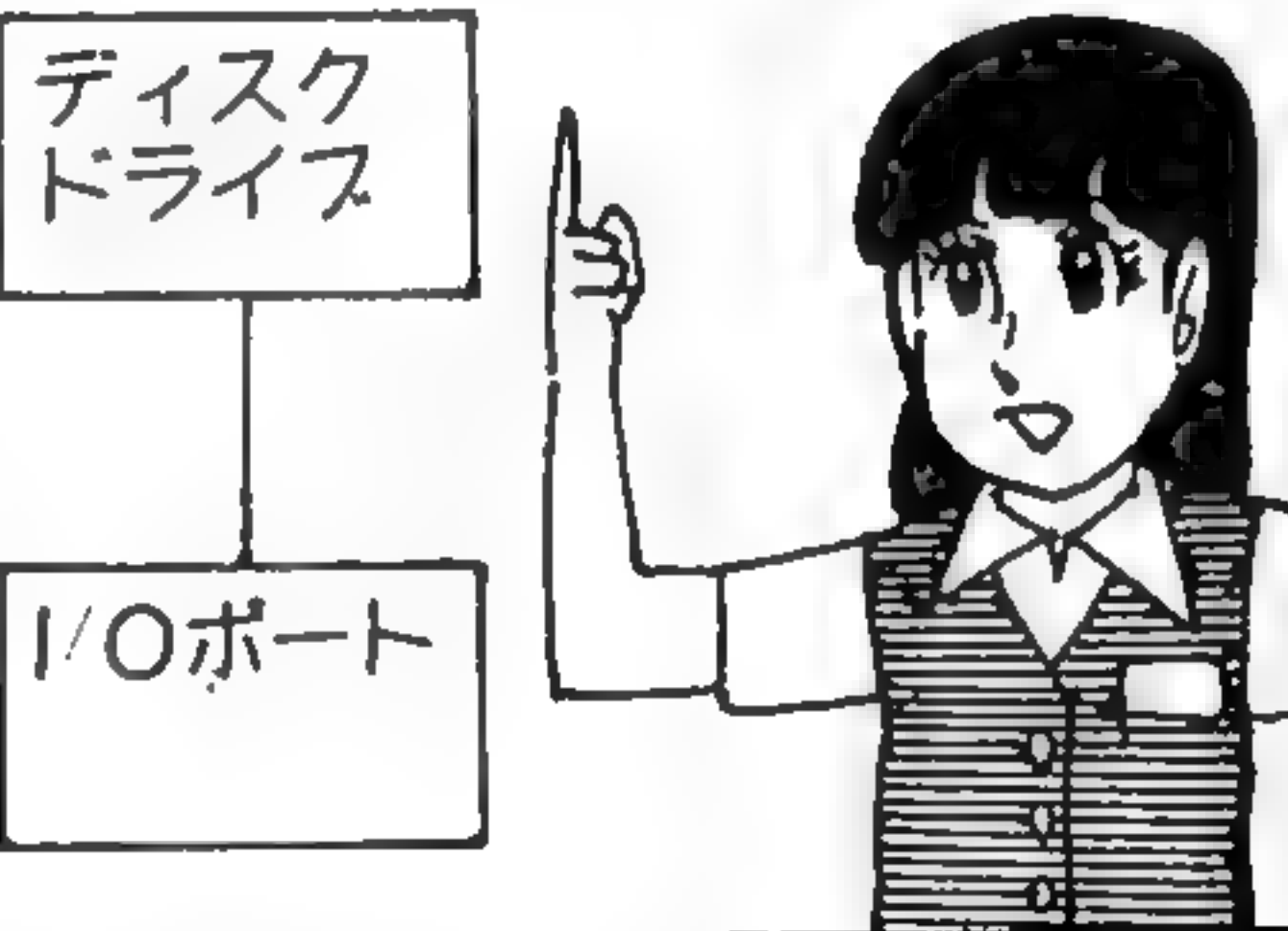
ばらばらでは、なんでこんな文があるのかわかりません。



次にCLOSE文がくるまではディスクドライブ#2がOPENされたままになっているのです。

ディスクドライブ

I/Oポート



FIELD文は何回でも使うことができます。FIELD文はその変数と長さの割当表を作るようなものでデータの移動はありません。

FIELD #1, 20 AS A\$, 30 AS B\$, 10 AS C\$  
FIELD #1, 40 AS G\$, 50 AS H\$  
FIELD #1, 128 AS J\$, 128 AS K\$

	1			2		3	
変数	A\$	B\$	C\$	G\$	H\$	J\$	K\$
バイト	20	30	10	40	50	128	128

バッファに変数とその長さを割り当てることになります。

FIELD #1, 20 AS A\$, 30 AS B\$, 10 AS C\$

A\$に20バイト割り当てる

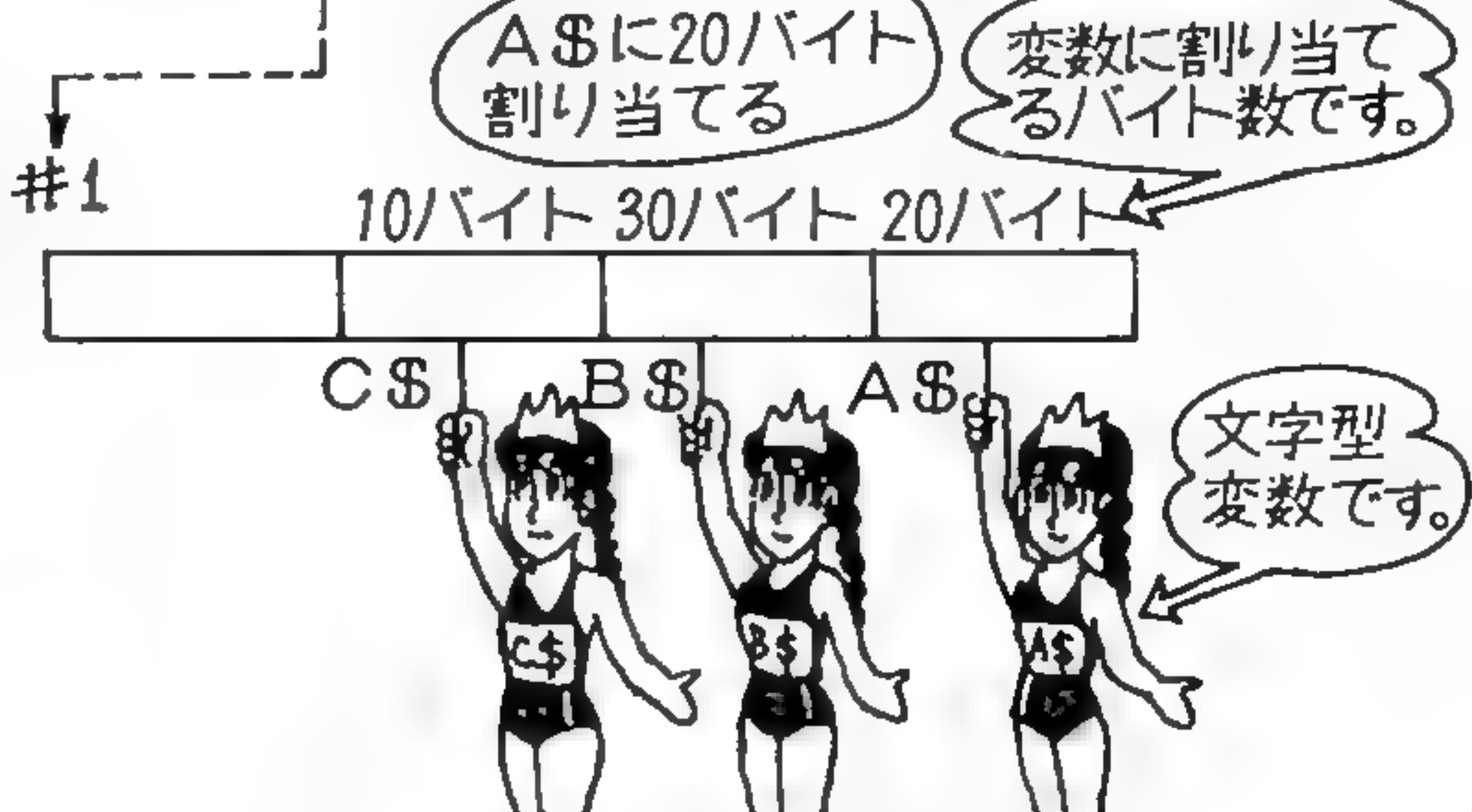
変数に割り当てるバイト数です。

#1

10/バイト 30/バイト 20/バイト

C\$ B\$ A\$

文字型変数です。





LとRは左と右のこと、割当てたバイトの左右どちらかに詰めてデータをバッファに移します。

LSET A\$ = P\$

← 左詰め

TAKUTO
--------

↑ 14文字文のアキ

TAKUTO
--------

Ⓢ P\$

プログラムで扱うデータをバッファに移す命令がLSET文とRSET文の2つです。

RSET

LSET

FIELD文によるバッファへの変数割り当てが完了したら、次はそれら変数の中身をバッファに移さねばなりません。

#1

20

P\$のデータ  
TUKUTO

FIELD

A\$	B\$	C\$
20	30	10

↑ A\$を使う

Ⓢ P\$

そこで数値をデータとしたい時にはまず数値を文字に変換しておく必要があります。

R\$

R

ここで注意が必要なのは、ランダムファイルはすべて文字型変数で取り扱われるということです。

P\$

Q\$

RSET B\$ = Q\$

B\$

A\$

#1

MASAYOSI	TAKUTO
----------	--------

28字文のアキ

↑ 右詰め

MASAYOSI
----------

Ⓢ Q\$

これで文字変換されたR\$の内容をバッファにセットできます。このR\$はC\$にセットしたいと思いますが、C\$の10バイトは本当は4バイトで十分なのです。

RSET C\$ = R\$

C\$	B\$	A\$
;°G■	MASAYOSI	TAKUTO

↑

;°G■
------

Ⓢ R\$

MKS\$関数では数値が4バイトの文字に変換されます。

R\$ = MKS\$(R)

;°G■
------

Ⓢ R\$

3.123
-------

Ⓢ R

この数値→文字変換がMKS\$関数です。

R\$ = MKS\$(R)

これは make single \$の略です。

PUT 1, 1が実行されると、ETUKOというファイルの1番目のセクタにバッファの内容が移されます。

#1

C\$	B\$	A\$
10/バイト	30/バイト	20/バイト

196バイトはアキ

ETUKO	;°G■	MASAYOSI	TAKUTO
-------	------	----------	--------

↑ ファイル名

1セクタ=256バイト

これでバッファに移したデータをフロッピーに書き込む用意ができました。PUT文を使います。

PUT文

PUT 1, n

#1

C\$	B\$	A\$
-----	-----	-----

↓

PUT 1, 1  
おのおの256バイト

ETUKO	1	2	...	n
-------	---	---	-----	---

セクタNo.



今、その範囲を越え、プログラムの中で使用するデータをランダムファイル形式でフロッピーに書き込めるようになったわけです。

	数学	国語	物理	化学
正義				
悦子				
拓人				

今までフロッピーに書き込んでいたのは、いろんな形式のプログラムだけでした。

SAVE "A"  
LOAD "A"

これでTAKUTO、MASA YOSHI、3.123という3つのデータをフロッピーに書き込んだわけです。

まずGET文でファイルからバッファにデータを移します。もちろん、その前に、ファイルはオープンされFIELD文でバイトの割当てが完了していません。

```
OPEN "2:ETUKO" AS #1
FIELD #1, 20 AS A$,
      30 AS B$, 10 AS C$
GET 1, 1
```

では、次にこの3つのデータをプログラムで取り出してみます。

ただ今のやり方では大量の数値をデータとして扱うのには不便な気がします。たとえば、数値(4バイト)を256バイトにどのように割り当てていったらよいかということです。へたをすると、長い長いFIELD文になってしまいます。

バッファ #1

4	4	4	4	4	...
A\$	B\$	C\$	D\$	E\$	...

この問題は、またあとで検討したいと思います。

この時、それぞれのデータがどんな形式で入っているのかわかっていないとうまくいきません。

あとは、バッファ #1 から必要なデータを取り出せばよいわけです。

C\$...10バイト 右詰め    B\$...30バイト 右詰め    A\$...20バイト 左詰め

;	6	MASAYOSI	TAKUTO
---	---	----------	--------

実は3.123という数字

LEFT\$    P\$

GET文はバッファの番号と、ファイルのセクタ(PUTしたセクタ)番号を指定するだけです。

1    2    3

ETUKO		
-------	--	--

GET 1, 1

#1

Q\$ = B\$ とすれば" Q\$ の中身は 22コのスペース(表示すると空白)と MASAYOSI (8文字)になります。

C\$の中の数値は4バイトの文字として入っていますのでRIGHT\$の他に、文字→数値変換のCVS関数も必要です。ひとつにまとめて書いてみました。

C\$

;	6
---	---

R = CVS(RIGHT\$(C\$, 4))

元の数字に戻す。

3.123

R

同様に、MASAYOSIはB\$30バイトの中に右詰めに入っているわけですから、RIGHT\$文で8文字取り出せばいいですね。

B\$

MASAYOSI
----------

Q\$ = RIGHT\$(B\$, 8)

MASAYOSI

Q\$

まず、TAKUTOはA\$20バイトの中に左詰めに入っています。これを取り出すにはLEFT\$文が便利です。A\$の左から6文字取り出すのは、LEFT\$(A\$, 6)と書き、これを別の変数に入れればよいのです。

A\$

TAKUTO
--------

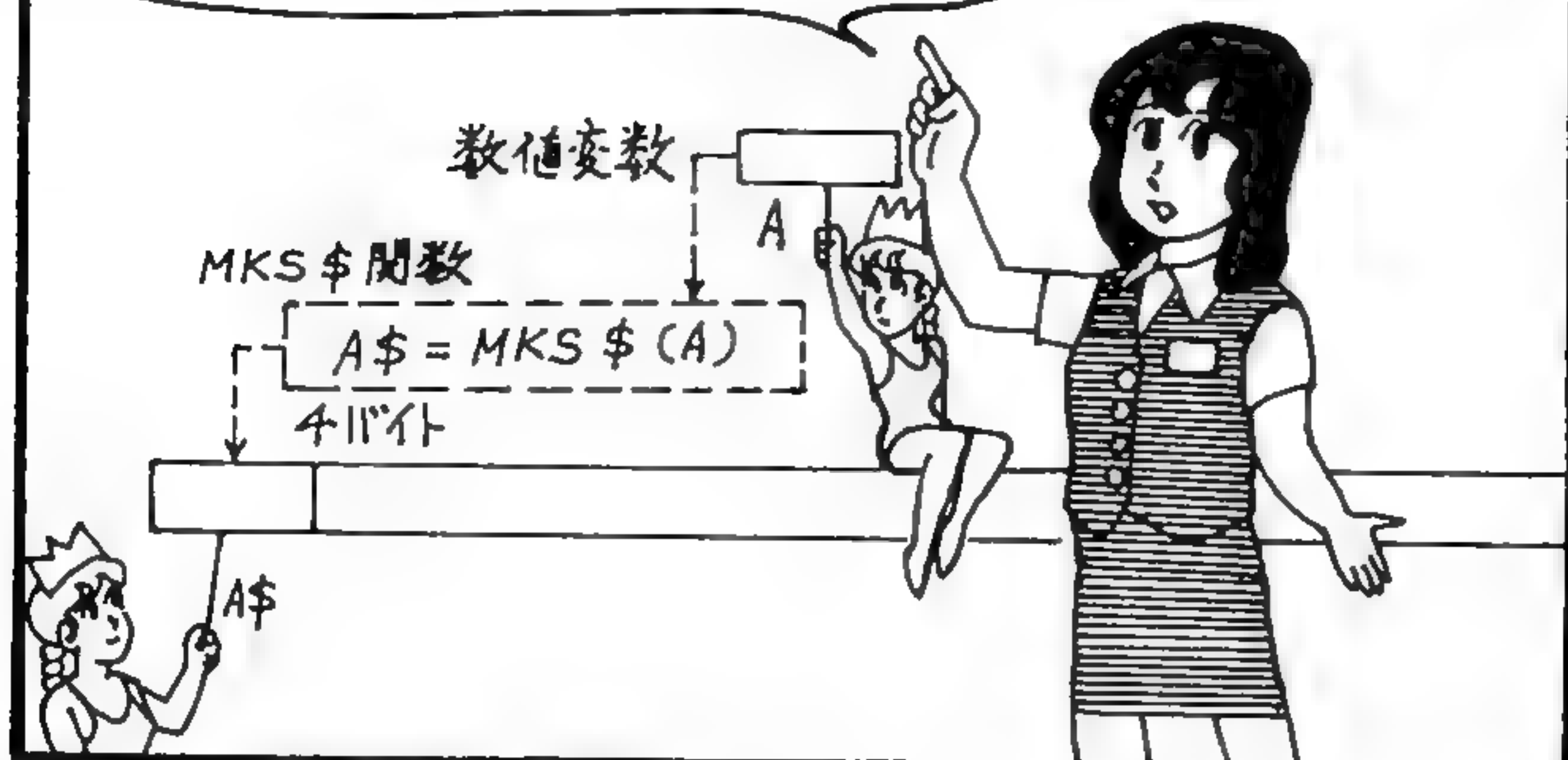
P\$ = LEFT\$(A\$, 6)

TAKUTO

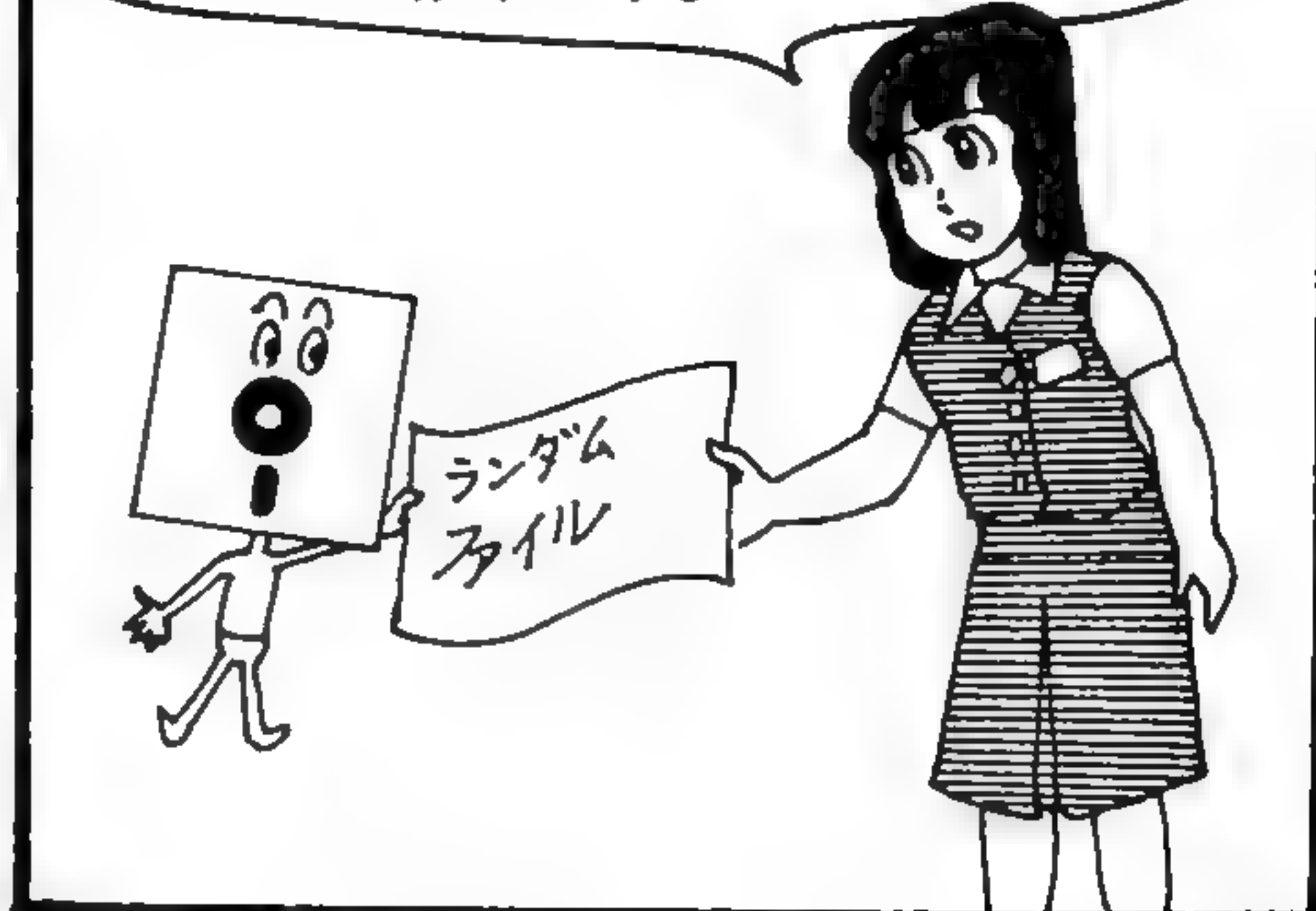
P\$



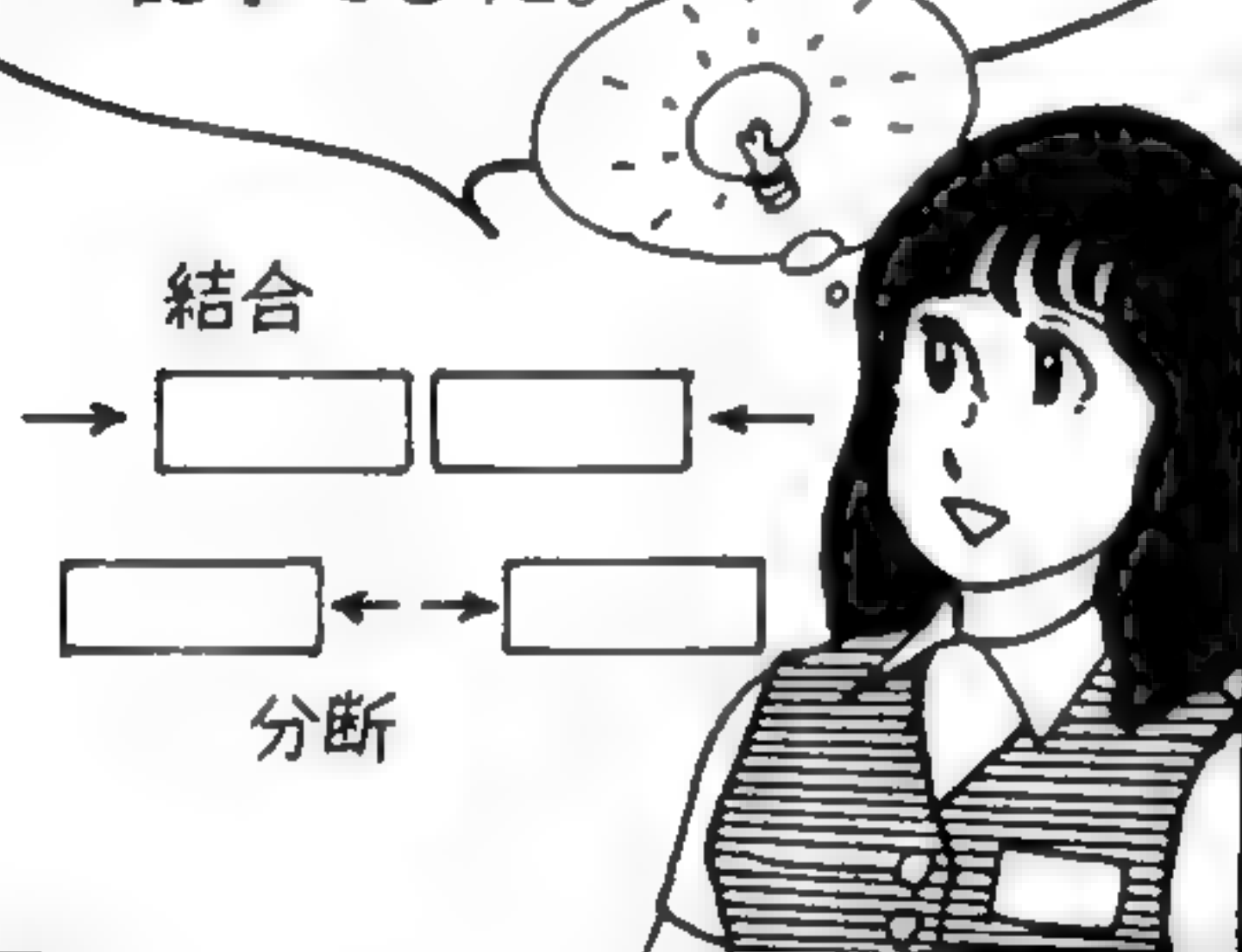
ところが、データが数値の場合は、FIELD文での変数割当てが問題です。



だいたいこんな感じで、フロッピーへのデータの読み書きができるわけです。



この時ふと思いついたのがポケコンのマニュアルに出ていた文字列の結合と分断の方法の記事でした。



しかし、64もの変数をどうやってFIELD文でバッファに振り当てたらいいのでしょうか。配列が使えなきゃ大変。

FIELD #1, 4 AS A\$  
4AS B\$ 4AS C\$  
4AS D\$ ...

64回もこれを書かなければならないなんて……。

このバッファに入れるのがすべて数値だとすると、256バイトを4バイトで割ればいいから、えーと

64個の数値が入れます。

数字  
15.689も  
4バイト  
で表わ  
せます。

↑  
4バイト

結合作業は配列変数を使わないと効果がありません。こんなくあいにやります。

```
DIM A$(32)
N=32
FOR I=1 TO N
  B$ = B$ + A$(I)
NEXT I
```

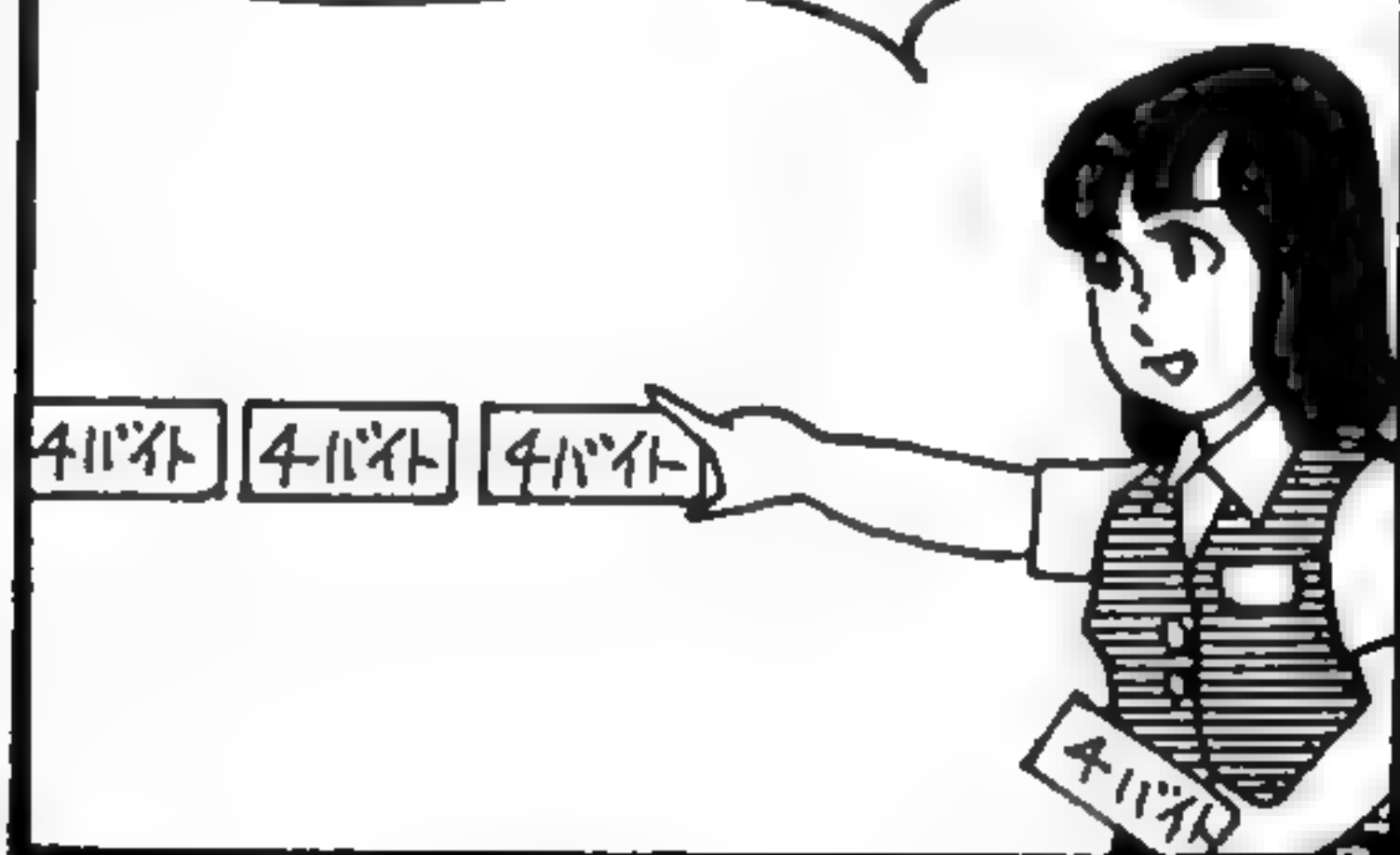
実行後B\$はA\$(1)がすべて4バイトとすると4×32バイトのなが〜い文字変数になってます。

文字列の結合はすごく簡単です。

B\$ = B\$ + A\$

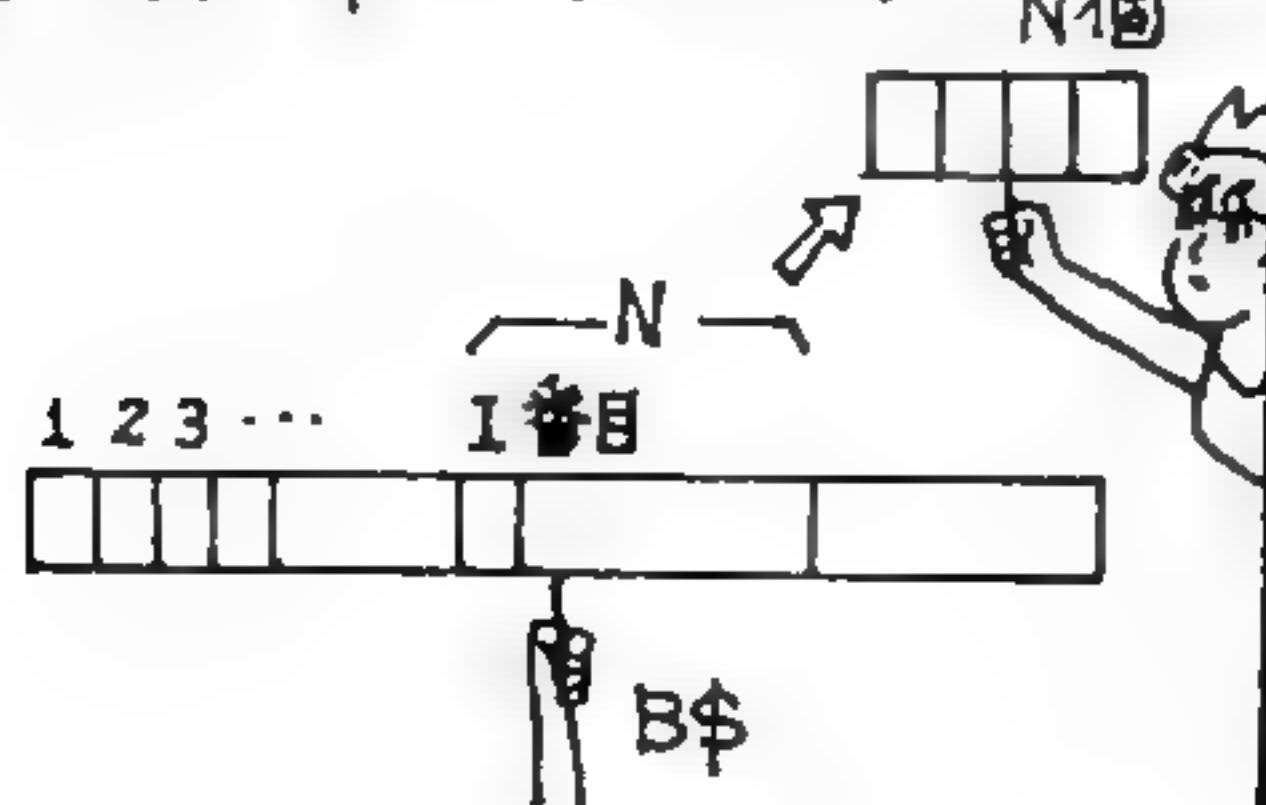
このようなたし算でOKなのです。

それは、バッファを入れる時は4バイトの文字列をいくつかくっつけて大きな文字列とします。そうすればバッファに定義する変数の数が少なくて済みます。

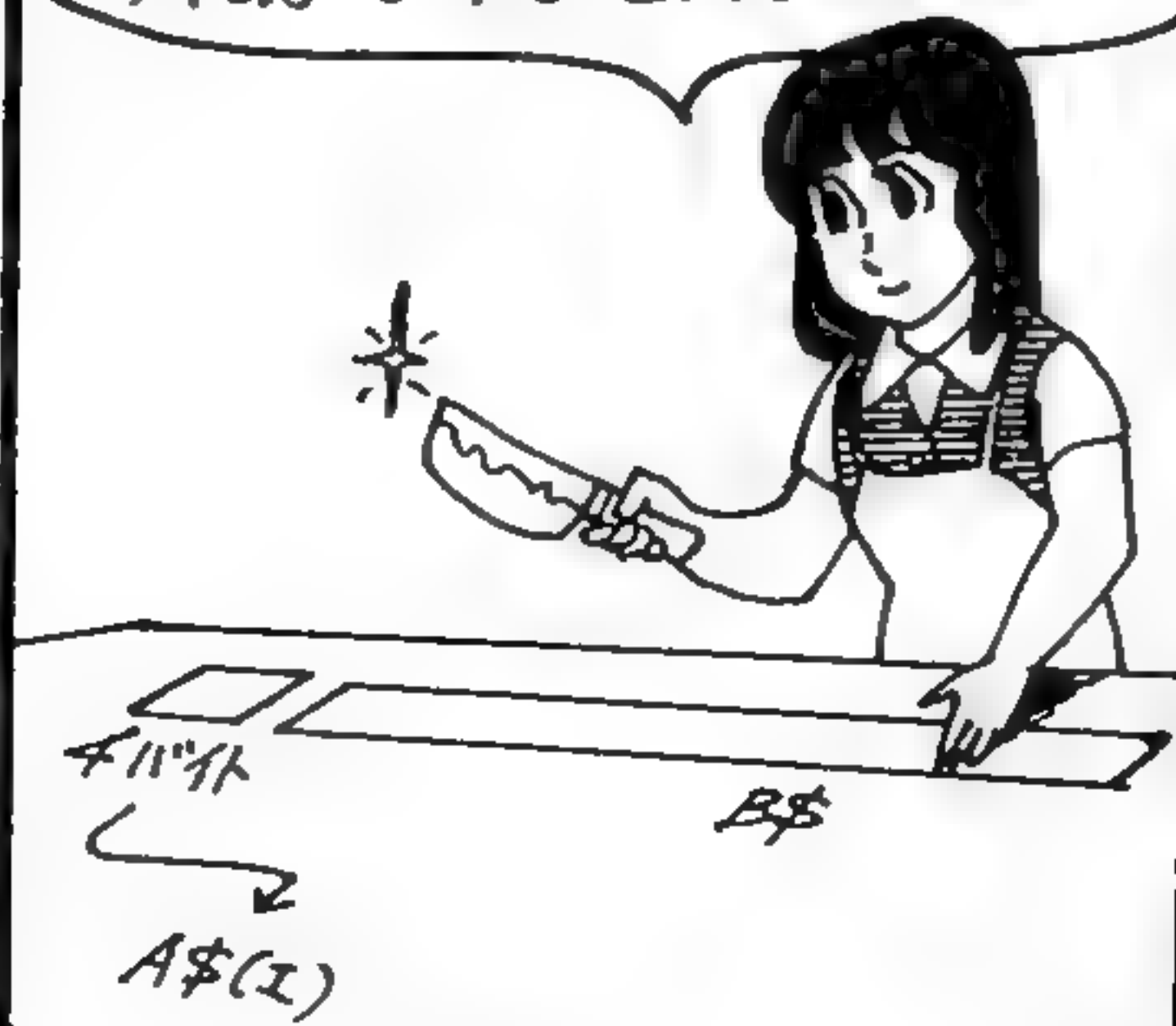


この時、包丁がわりに使わねばならないのがMID\$関数です。文字列の左がわ1番目からN個の文字を取り出すことができます。

A\$ = MID\$(B\$, I, N)

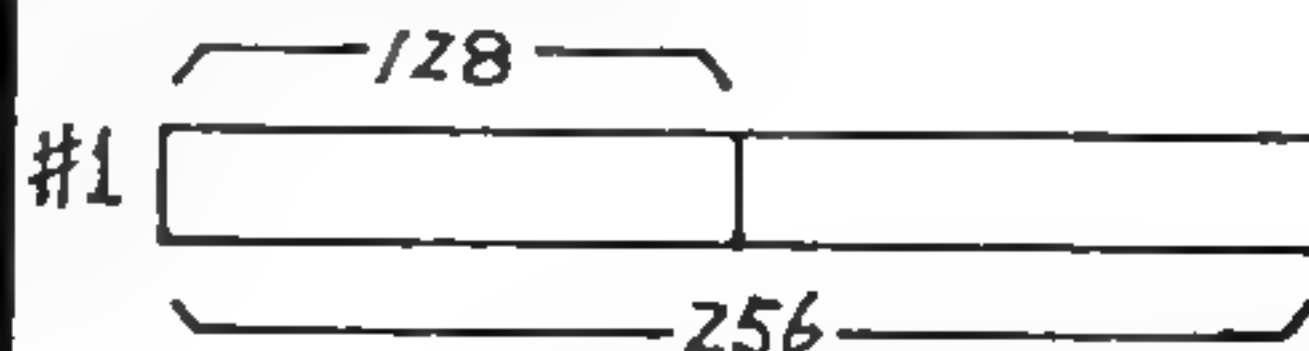


次に取り出す時は、このB\$を4バイトずつに切り出してA\$(I)に順番に戻してやらなければなりません。



このB\$ひとつでバッファの半分を占めることになります。

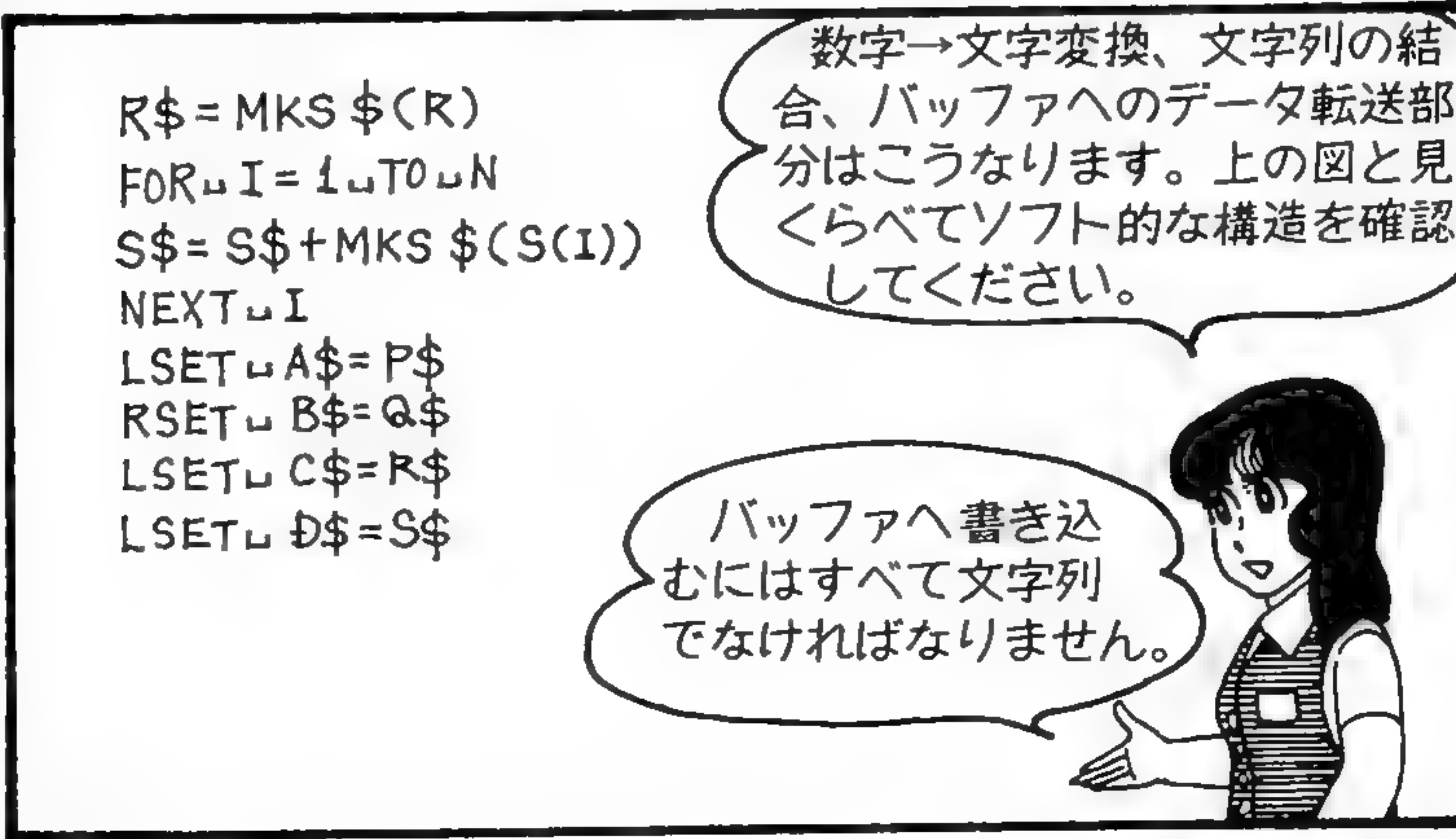
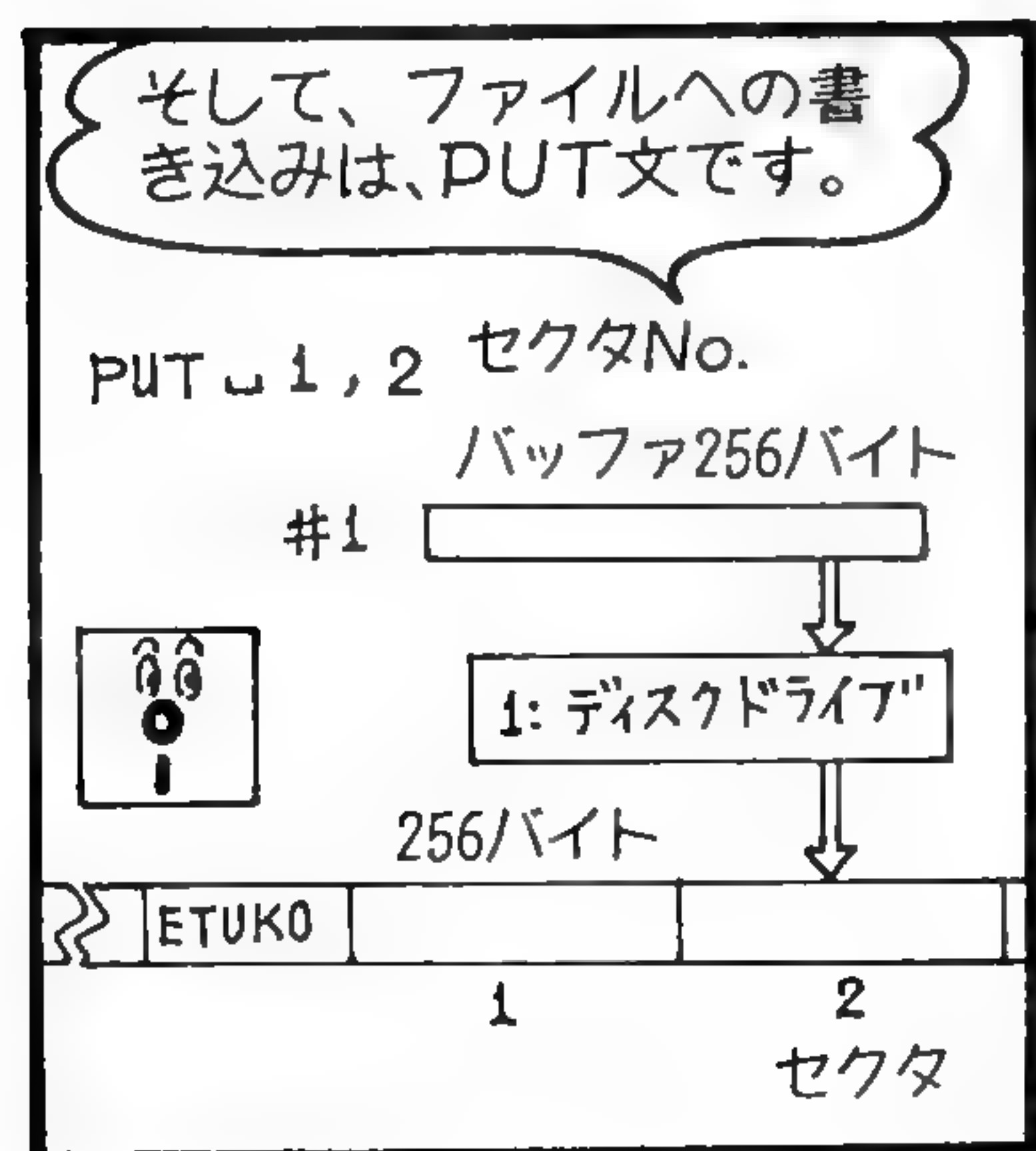
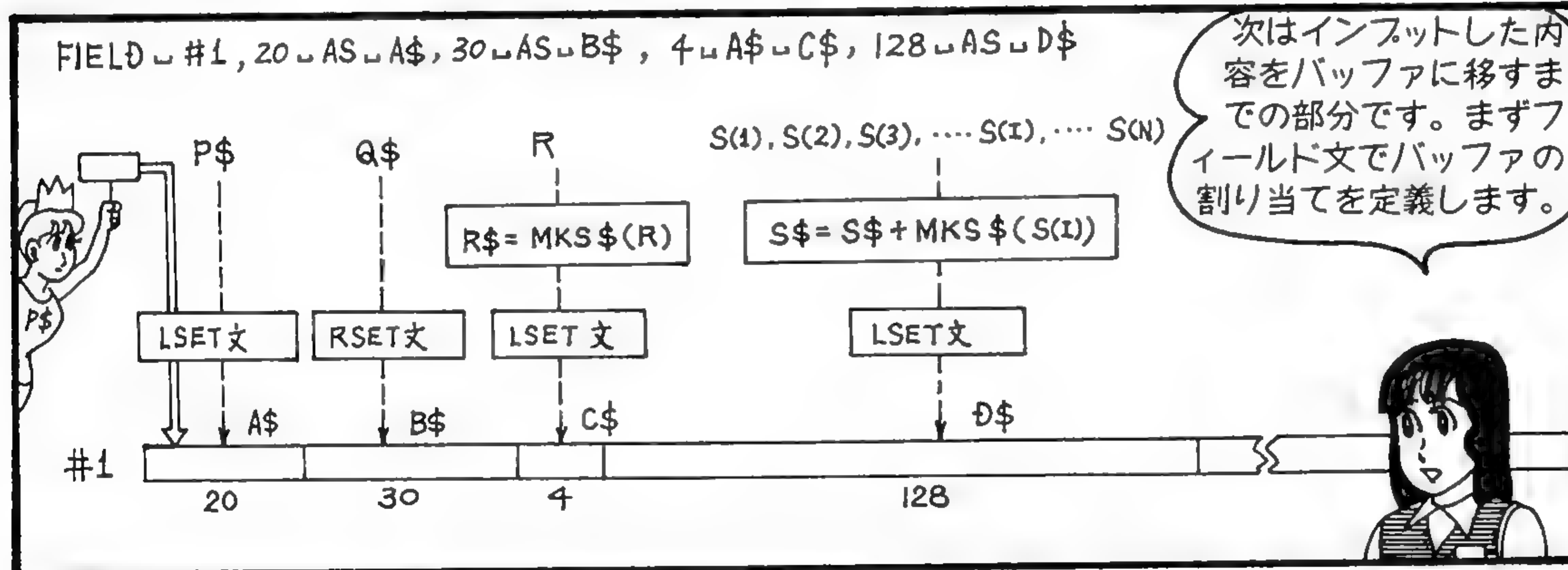
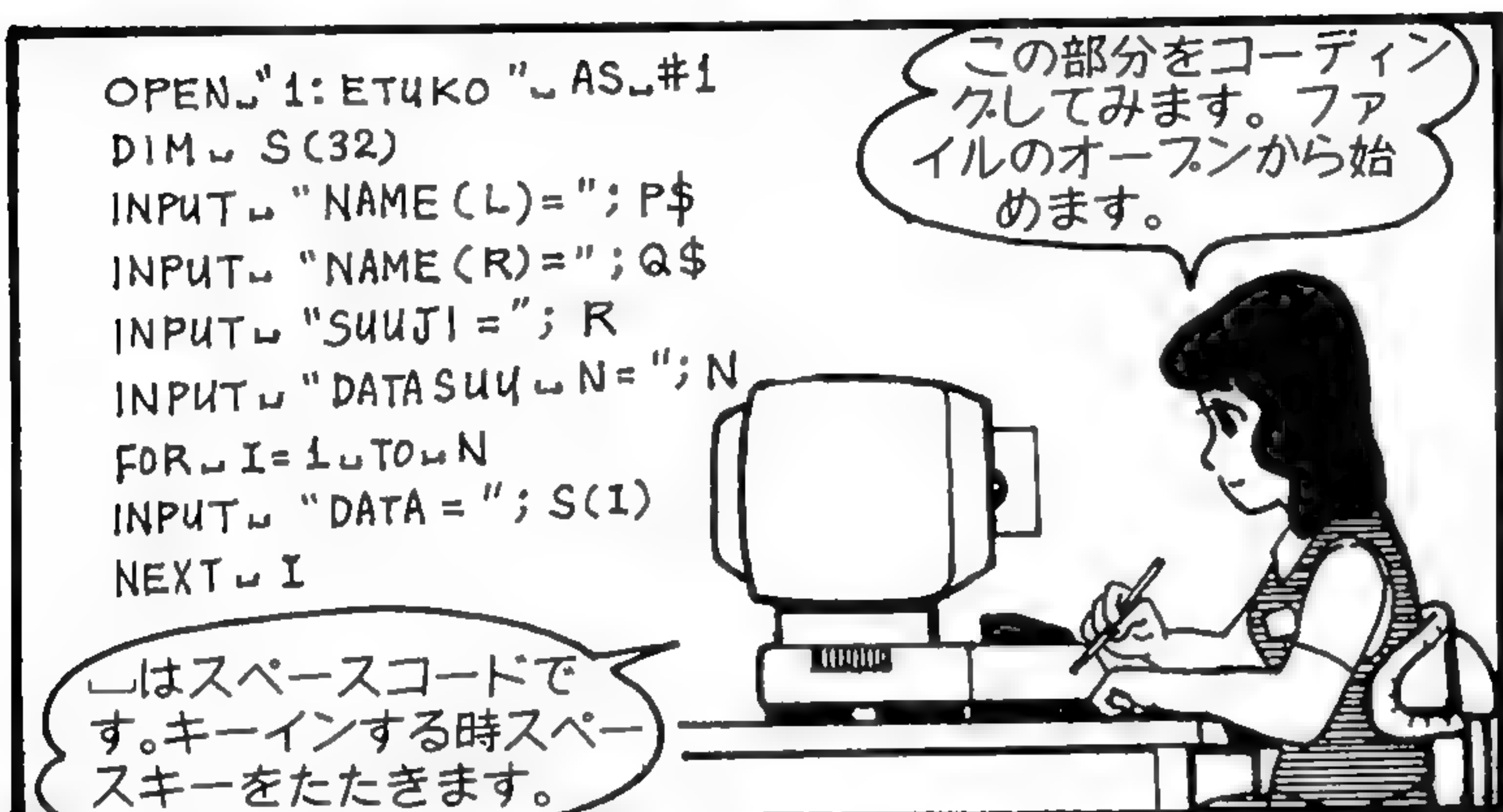
FIELD #1, 128 AS B1\$  
LSET B1\$ = B\$



B\$は128バイト以下でなければなりません。



テストプログラムの内容  
文字変数（左づめ、右づめ）  
と数値と配列値を一度フロッ  
ピーに書き込みます。  
次に使った変数をクリアして  
消えたかどうか確認の表示を  
します。  
次にフロッピーの内容を読み  
出してこれを表示してみよう  
というものです。





```

P$ = ""
Q$ = ""
R = 0 : R$ = MKS$(R)
FOR I = 1 TO N
  S(I) = 0
NEXT I
S$ = ""
LSET A$ = P$
RSET B$ = Q$
LSET C$ = R$
LSET D$ = S$

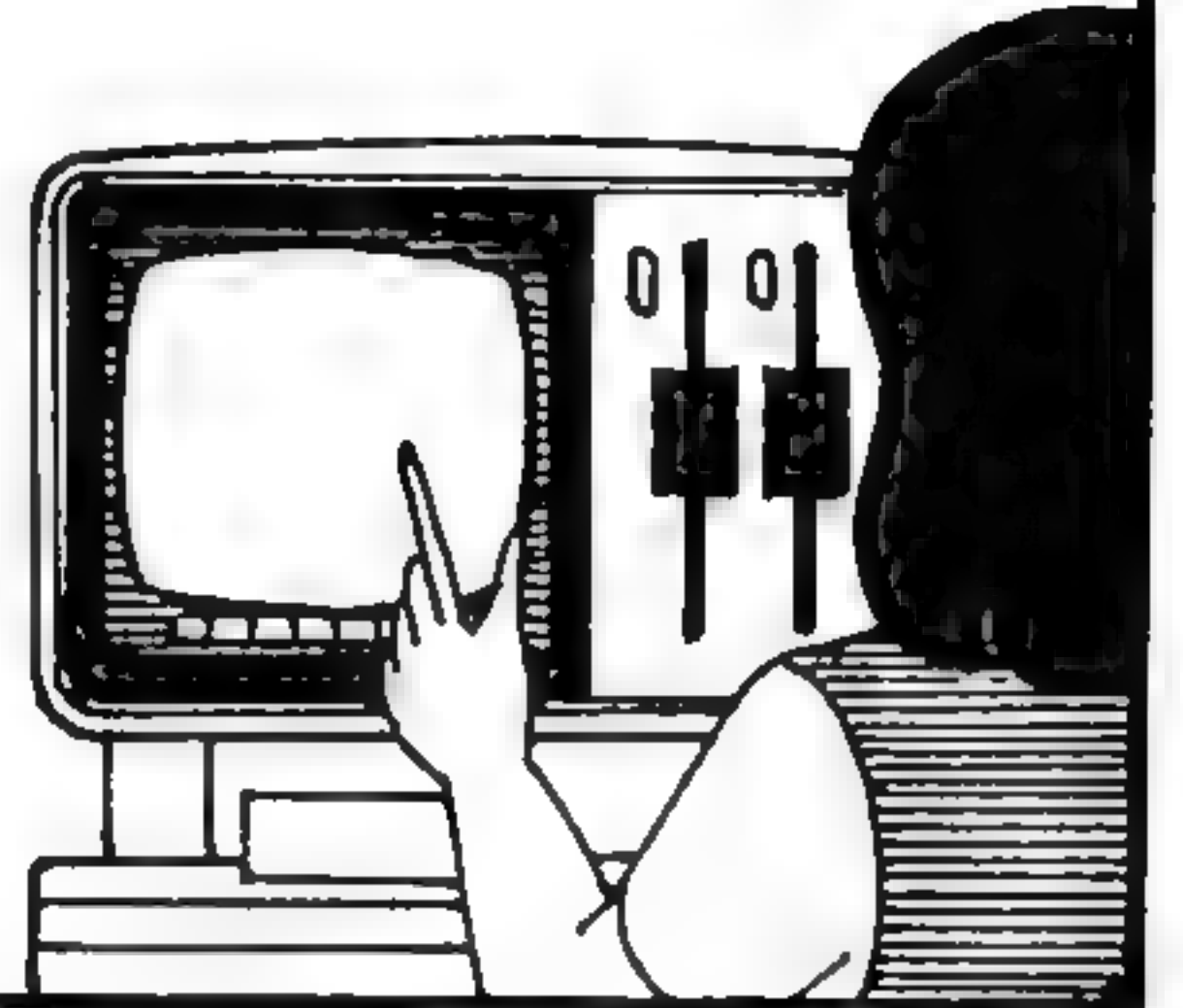
```

また、バッファの内容と使った変数の内容をクリアしておきます。ここでCLOSEして電源を切ってやり直せばこんなことしなくてもいいんですけどね。

文字列のクリアは ` ` です。

ここでインプットの時の表示内容などをクリアしておきましょう。CLS文です。

CLS

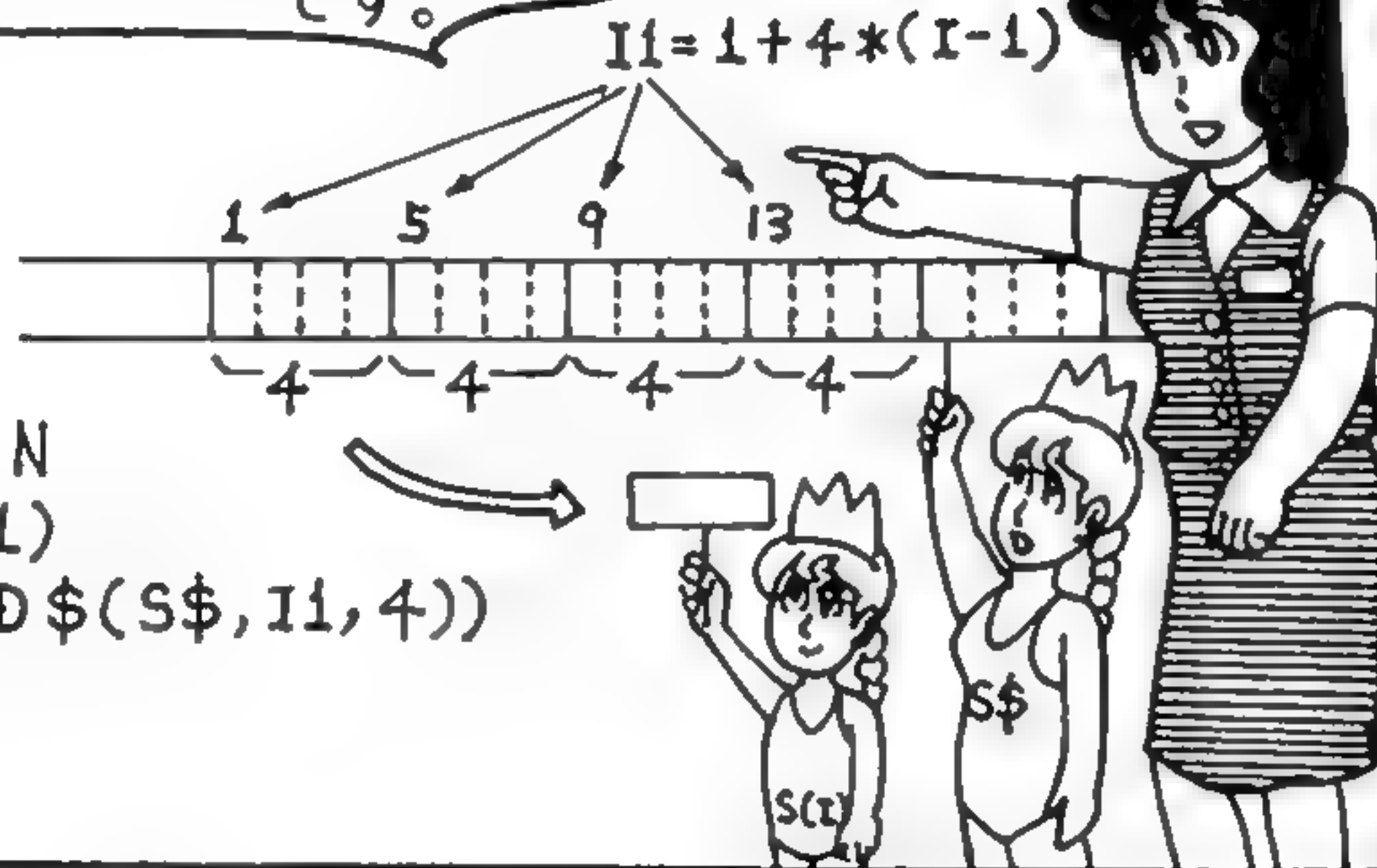


ここでのハイライトはS\$を4バイトずつに分解してS(I)に入れるところです。

```

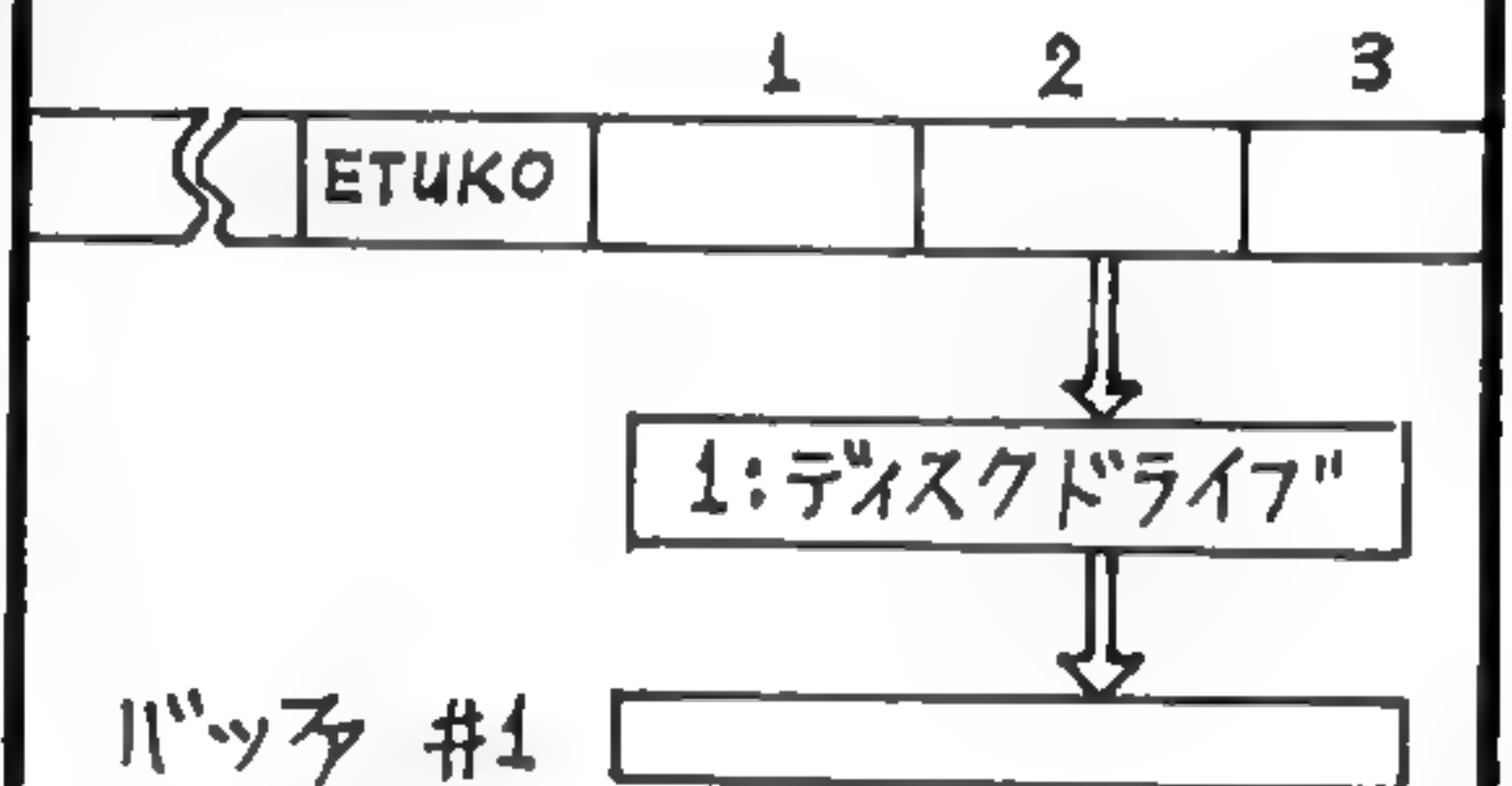
P$ = A$
Q$ = B$
R$ = C$
S$ = D$
FOR I = 1 TO N
  I1 = 1 + 4 * (I - 1)
  S(I) = C$ (MID$(S$, I1, 4))
NEXT I

```



次にGET文でファイルからバッファヘータを読み込みます。

GET 1, 2



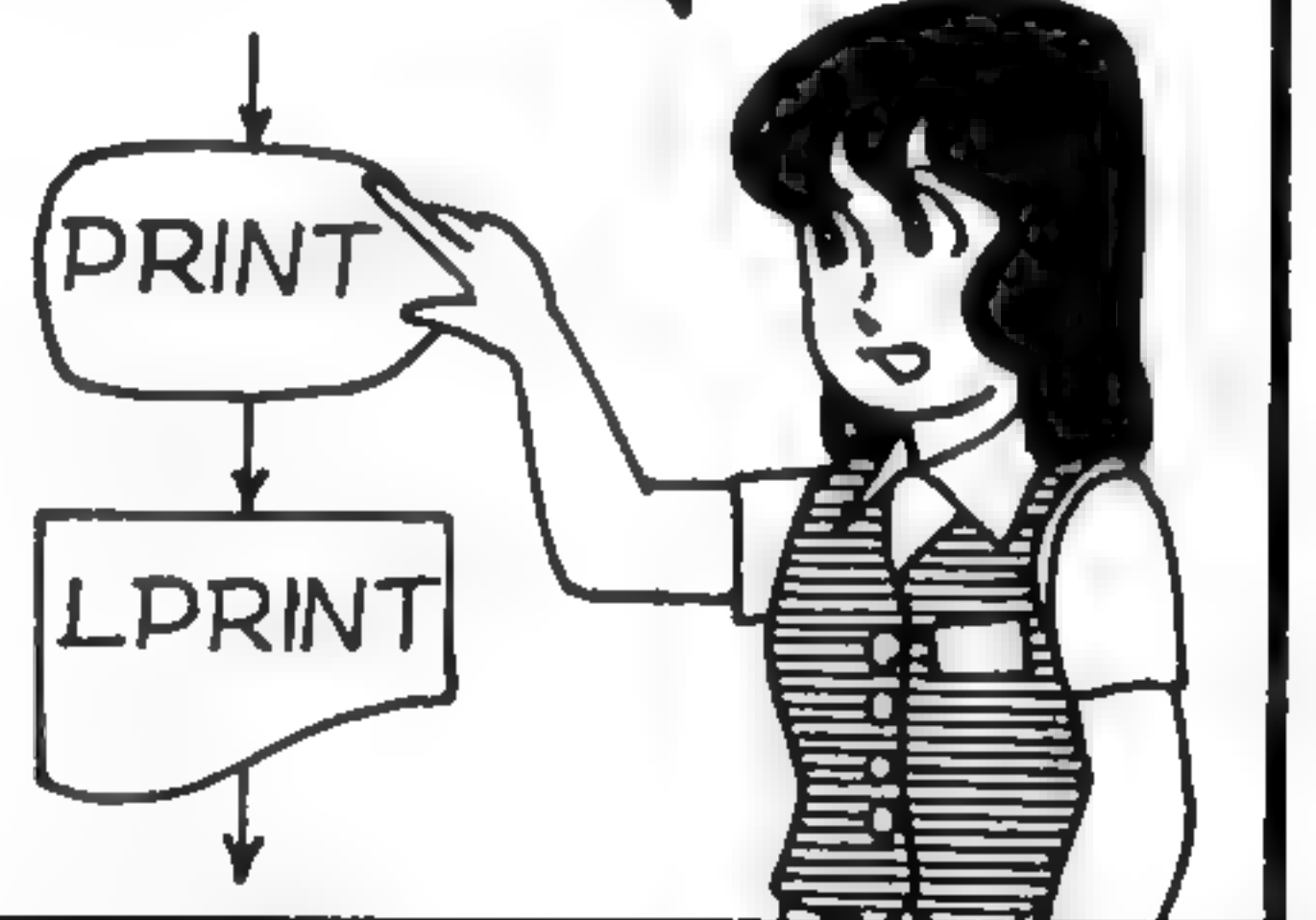
```

PRINT "NAME(L)="; P$; LPRINT "NAME(L)="; P$
PRINT "NAME(R)="; Q$; LPRINT "NAME(R)="; Q$
PRINT "SUUJI="; R; LPRINT "SUUJI="; R
PRINT "DATASUUUN="; N; LPRINT "DATASUUUN="; N
FOR I = 1 TO N
  PRINT USING "S(###)=####.##"; I; S(I)
  LPRINT USING "S(###)=####.##"; I; S(I)
NEXT I

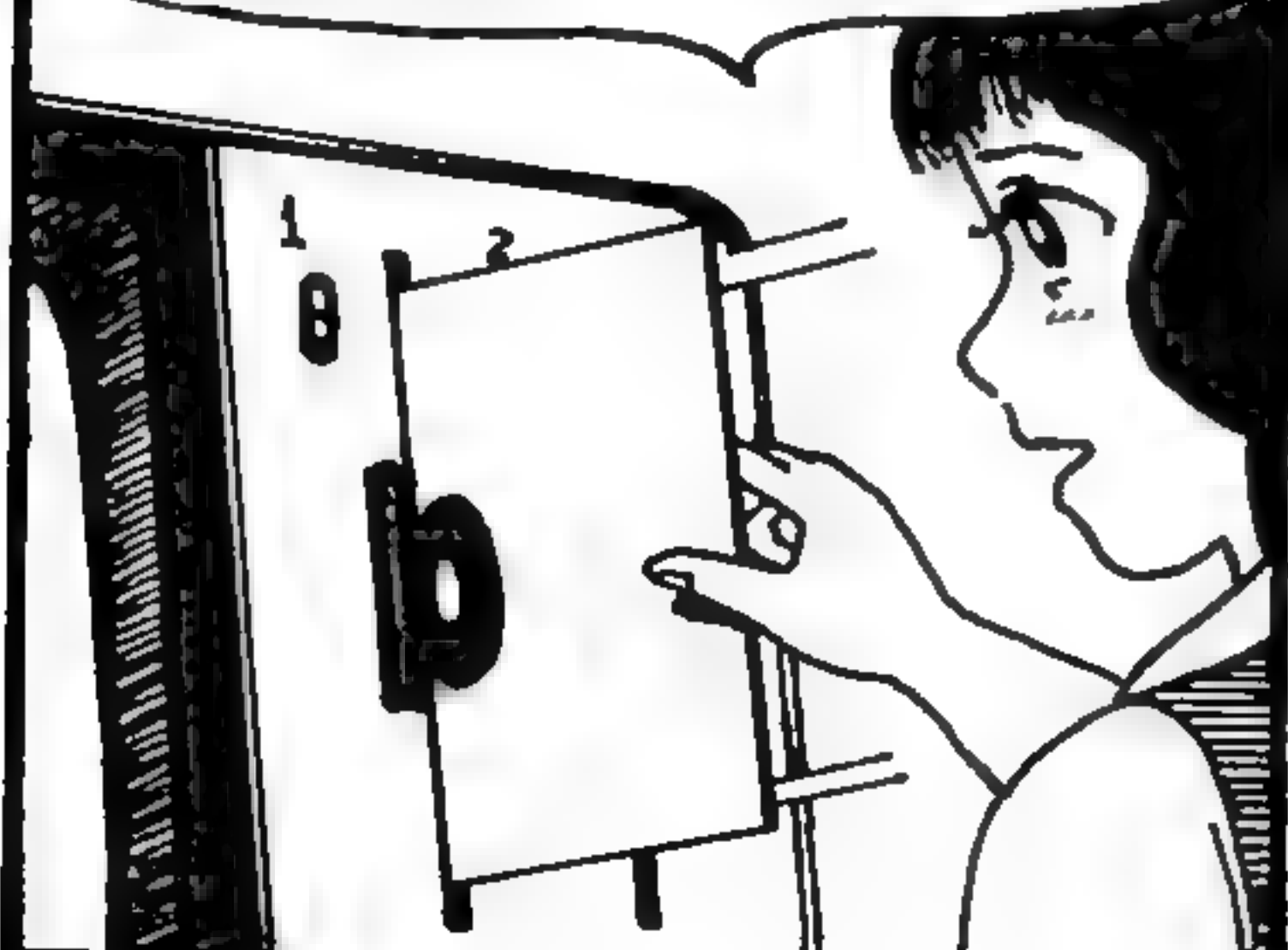
```

ポケコンの場合とUSINGフォーマット指定が違ってきます。

あとはPRINT文で表示すればうまくいったかどうかわかります。ついでですからLPRINT (エルフプリント)文で印字もします。



では、やってみましょう。まずFDD(フロッピーディスク)をドライブ1に入れます。C/P/Mではドライブ1はAと表現されます。2はBです。



あとは各ステートメントにNo.をつければコーディングの完了です。No.の次に'のあるのは注釈行で、プログラムの実行には関係ありません。

```

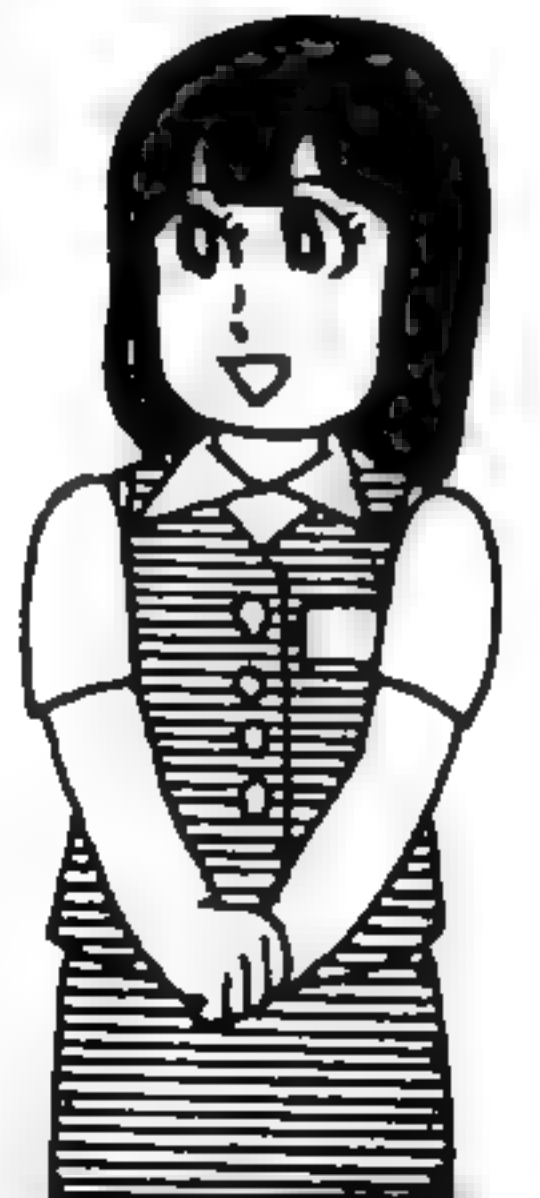
10 ' SAVE CODE
20 ' AN EXAMPLE OF
30 ' DIM S(32) : ' NUNDSAI
40 ' OPEN "1:ETUKO" AS #1

```

このフランクコードもキーインしないと文法エラーになります。

これで終わりですからファイルをクローズします。そして、ENDです。

CLOSE #1  
END





これでCP/Mが立ち上がります。表示のA>はCP/Mのプロンプト状態を示し、Aはドライブ1のモードを示しています。

画面にクリーンの横線が表示されたら最後まで挿入し、フタを閉じます。

フロッピーは途中まで入れ、そのまま電源スイッチをONにします。

まず、最初の文番号として、10が表示されます。ステートメントを入力しRETURNするたび10ずつ増えます。

```
OK
AUTO
10 ' AN EXAMPLE OF
20 ' フロッピー - SAVE コード
30 CLS
```

AUTOの場合、最初のリンクは自動的にとられます。

次は、プログラムのインプットですが文番号発生機能のAUTOコマンドを使うと便利です。

```
OK
AUTO
```

AUTO RETURN

次にOBASICRETURNでBASICのインタプリタが立ち上がります。このBASICはOSであるCP/Mのもとで走ることになります。プロンプト状態はOKで示されます。

```
OK
```

OBASIC RETURN

AUTOを解除したい時はCAN キャンセルキーを使います。

```
580 END
590
```

CAN

プログラム上よく使うINPUTやPRINTはCOMDキーとIやPの同時使用で一度に入力されるようになっています。たとえばINPUTならCOMDとIを同時に押せばいいのです。

```
40 INPUT
```

COMD I

文番号までいちいちキーインするとスペースの1まで必要です、なかなかめんどくさいです。

次にINPUT文がありますので、NAME(L)=で示されるガイダンスと共に?が表示されます。

```
NAME(L)=?
```

このプログラムには最初にOPEN文がありますので、ドライブ1が起動します。

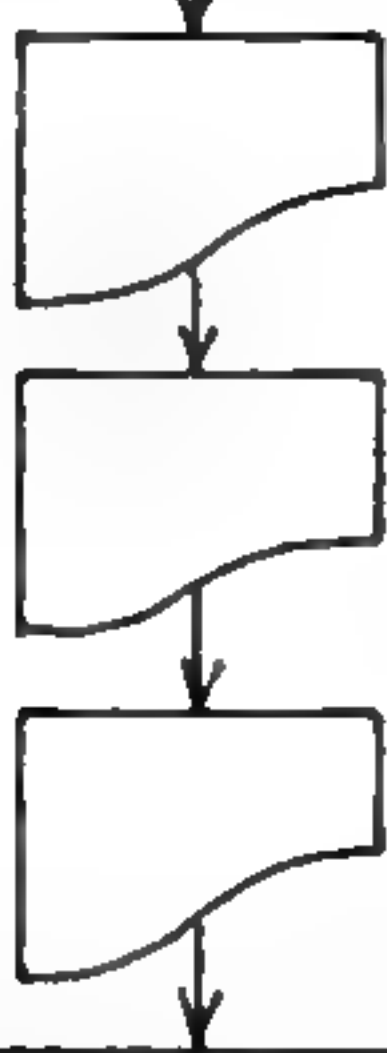
プログラムの実行はRUNコマンドです。ファンクションキーにRUNと同じ機能のものがあるはずですが、もちろん、ひとつひとつキーを押していてもいいのです。

```
RUN
```

RUN RETURN



表示と印字は、インプット内容の確認、クリアの確認、ファイルからの読出し確認のため3回することにした。



NAME(L)=? TAKUTO タクト  
NAME(R)=? MASAYOSI マサヨシ  
SUUJI(####.##)=? 123.45  
DATASUUN(Max.32)=? 3  
DATA(####.##)=? 111.11  
DATA(####.##)=? 222.22  
DATA(####.##)=? 333.33

表示された項目のデータを入れていきます。

この例ではN=3  
なので3つ目のデータを入  
れ終わった時にプログラ  
ムが次に進みます。

2 2 RETURN  
3 3 RETURN

(( ))

次の360番のGOSUB文では  
クリアされた内容がプリント  
されます。Nはプログラム上  
クリアできません。

DATA(3)= 333.33

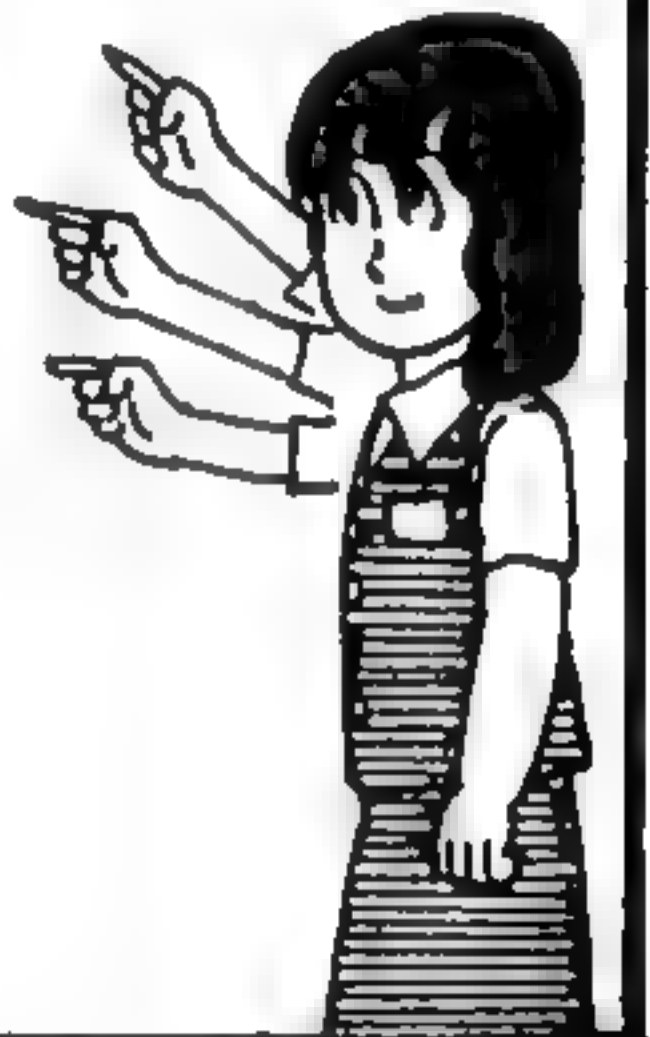
NAME(L)=  
NAME(R)=  
SUUJI= 0  
DATASUUN= 3  
DATA(1)= 0.00  
DATA(2)= 0.00  
DATA(3)= 0.00

最初の135番のGOSUB文で  
はインプットの内容がそのま  
ま出力されます。

NAME(L)= TAKUTO タクト  
NAME(R)= MASAYOSI マサヨシ  
SUUJI= 123.45  
DATASUUN= 3  
DATA(1)= 111.11  
DATA(2)= 222.22  
DATA(3)= 333.33

同じ変数の内容確認なので、  
表示、印字ルーチンをサブル  
ーチンにしました。

135 GOSUB 490  
:  
360 GOSUB 490  
:  
470 GOSUB 490  
:  
GOTO 570  
490 PRINT  
:  
570 RETURN  
END



これでなんとなくランダムファ  
イルが使える気がしてきました。  
自由自在に使いこなせるのは時  
間の問題です。



そして、ファイルをCLOSE  
し、END文でプログラム終了  
です。



最後に470番でファイルから読  
んだ内容を出力します。

DATA(1)= 0.00  
DATA(2)= 0.00  
DATA(3)= 0.00

NAME(L)= TAKUTO タクト  
NAME(R)= MASAYOSI マサヨシ  
SUUJI= 123.45  
DATASUUN= 3  
DATA(1)= 111.11  
DATA(2)= 222.22  
DATA(3)= 333.33

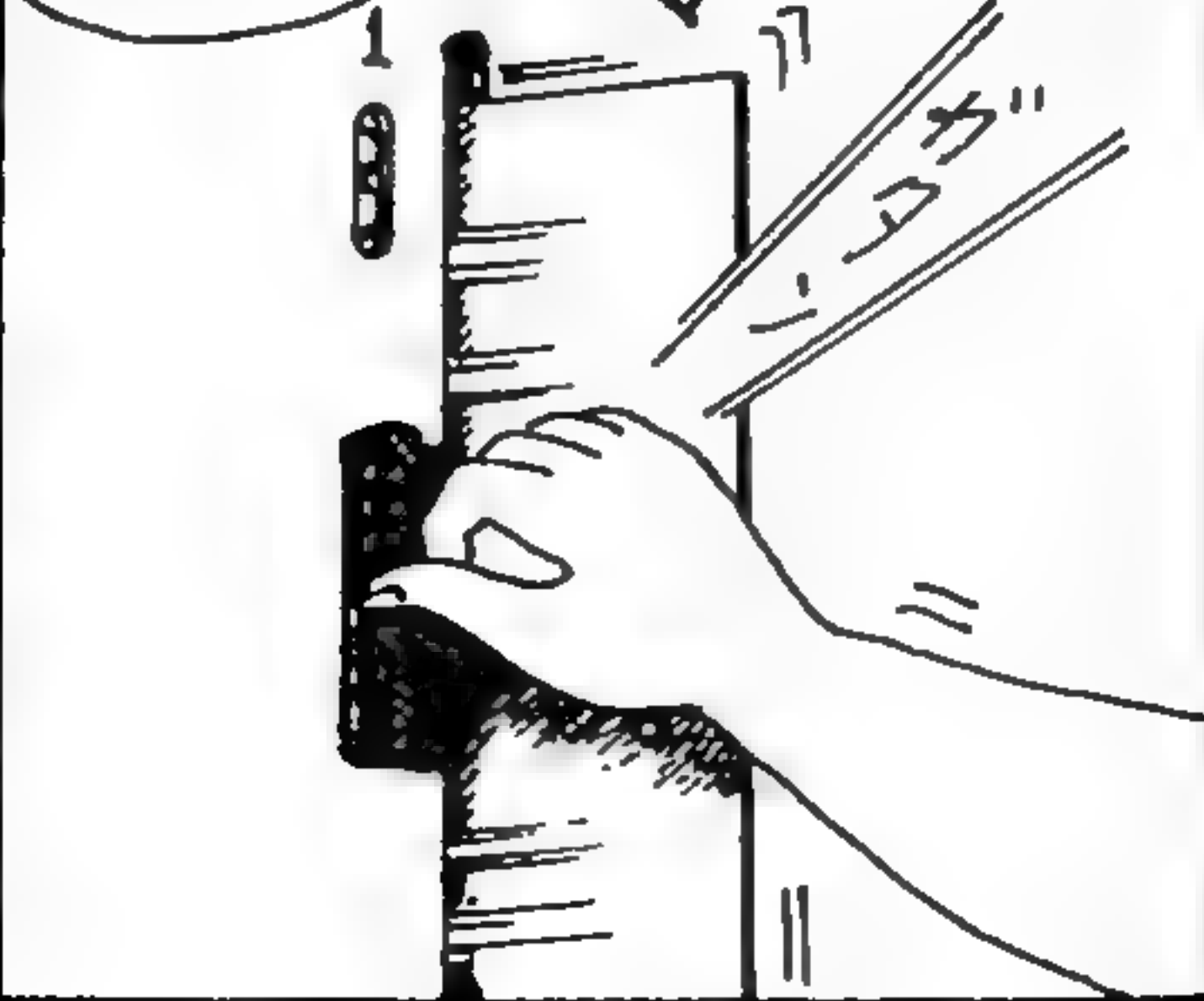
ここに  
RとLの  
差が出て  
います。

最初ETUKO 1でSAVEし  
たのですがあれこれ修正してい  
るうちに改訂9になりました。

ETUKO1  
ETUKO2  
:  
ETUKO8  
ETUKO9



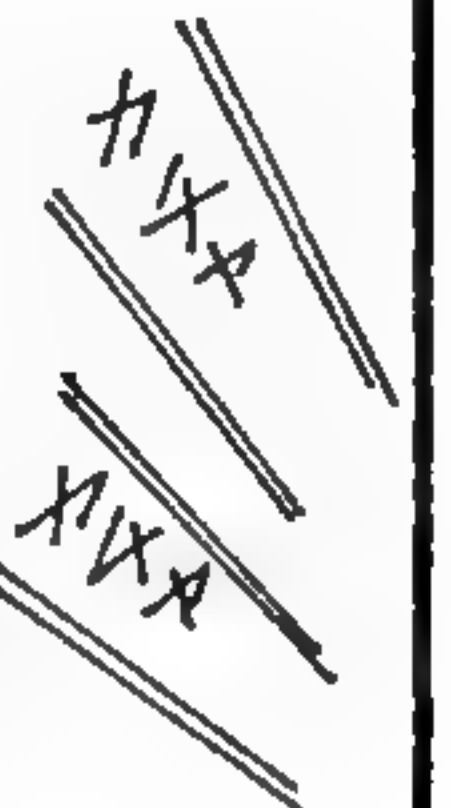
FDIはRESET RETURN  
と入れすべてのファイルをCL  
OSEしてから取り出してくだ  
さい。



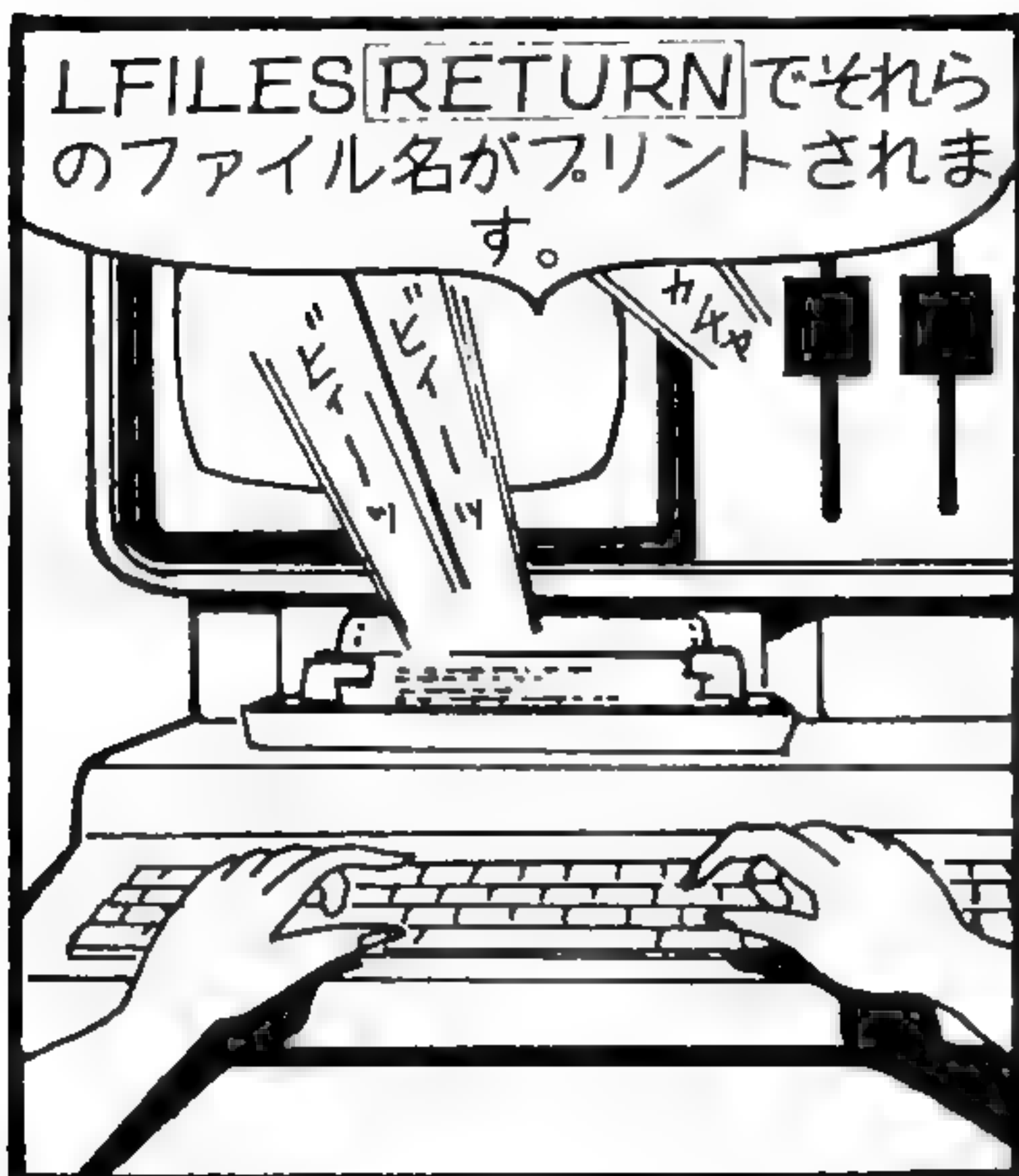
プログラムは一応FDDにしま  
っておきましょう。

SAVE "ETUKO9"

プログラムの内容を変  
更したい時は名前も変  
えてSAVEした方が  
いいでしょう。



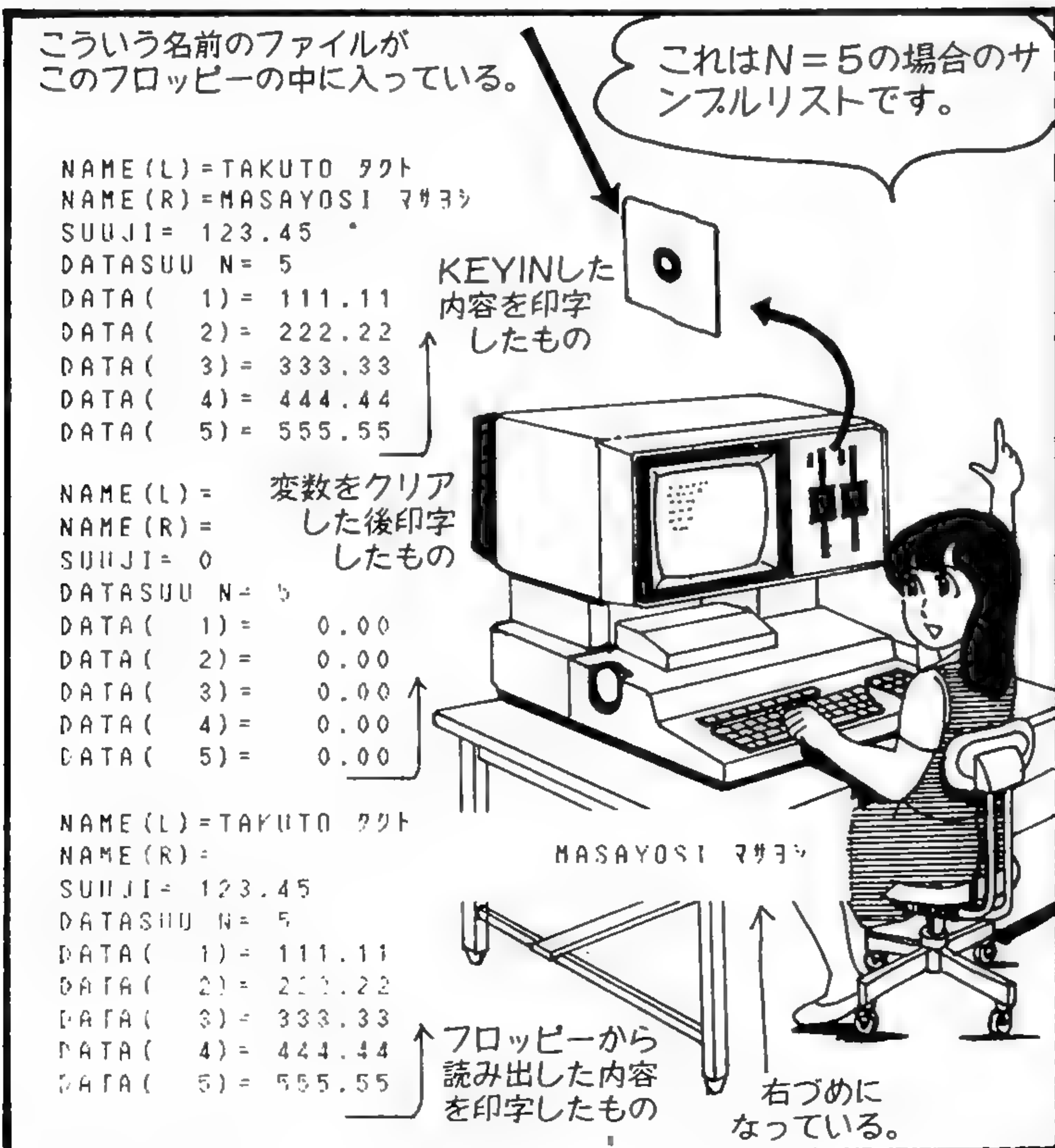




MOVCFM	.COM	PIP	.COM
ASM	.COM	DDT	.COM
DUMP	.COM	DUMP	.ASM
DISKDEF	.LIB	FDDUTY	.COM
OBASIC	.COM	IFINSTL	.COM
ETUKO	.	ETHUKO	.
ETUKO4	.	ETUKO5	.
ETUKO9	.		

SUBMIT	.COM	XSUB	.COM
LOAD	.COM	STAT	.COM
BIOS	.ASM	CBIOS	.ASM
KANJI	.COM	IF20T030	.DOC
IFCONFIG	.COM	2	.
ETUKO1	.	ETUKO2	.
ETUKO6	.	ETUKO7	.

ED	.COM
SYSGEN	.COM
DEBLOCK	.ASM
IFDEM01	.BAS
A	.
ETUKO3	.
ETUKO8	.





このプログラムリストは'で始まる注釈行を入れたためにややこしく見えますが、コーディング時に説明した内容とそう変わりません。'はREMの略記で、プログラムの実行には関係ありません。プログラムのメモ書きとしてプログラマが入れておくものです。

' = REM



プログラム内容を表示させるのはLIST RETURN プリントはLLIST RETURNです。



```

10 'AN EXAMPLE OF RANDAM FILES HANDLING
20 ' フロッピー - SAVE コード' A ETUKO9
30 CLS
40 DIM S(32)
50 OPEN "1:ETUKO" AS #1
60 FIELD #1,20 AS A$,30 AS B$,10 AS C$,128 AS D$
70 INPUT "NAME(L)=";P$: 'ヒタリツメ モシ"
80 INPUT "NAME(R)=";Q$: 'ミキツメ モシ"
90 INPUT "SUUJI(####.##)=";R: 'スウシ"
100 INPUT "DATASUU N(Max.32)=";N
110 FOR I=1 TO N
120 INPUT "DATA(####.##)=";S(I)
130 NEXT I
135 GOSUB 490:PRINT:LPRINT
140 R$=MK$$(R)
150 FOR I=1 TO N
160 S$=S$+LEFT$(MK$$(S(I)),4): ' 4バイト フォーマット
170 NEXT I
180 LSET A$=P$: ' バッファ #1 にセット
190 RSET B$=Q$
200 LSET C$=R$
210 LSET D$=S$
220 PUT 1,2: ' バッファ #1 から フロッピー - へ
230 ' ETUKO タイトル 2 セクタ に書き込み
250 P$="": ' リセット ルーチン
260 Q$=""
270 R=0:R$=MK$$(R)
280 S$=""
290 FOR I=1 TO N
300 S(I)=0
310 NEXT I
320 LSET A$=P$
330 RSET B$=Q$
340 LSET C$=R$
350 LSET D$=S$
360 GOSUB 490:PRINT:LPRINT: ' フォーマット (リセット)
370 GET 1,2: ' ETUKO タイトル 2 セクタ から
380 ' バッファ #1 に読み出し
390 P$=A$
400 Q$=B$
410 R$=C$:R=CVS(R$)
420 S$=D$
430 FOR I=1 TO N
440 I1=1+4*(I-1)
450 S(I)=CVS(MID$(S$,I1,4)): ' 4バイト にフォーマット
460 NEXT I
470 GOSUB 490:GOTO 580: ' フォーマット (3ミタリ カタリ)
480 ' フォーマット ルーチン
490 PRINT "NAME(L)=";P$:LPRINT "NAME(L)=";P$
500 PRINT "NAME(R)=";Q$:LPRINT "NAME(R)=";Q$
510 PRINT "SUUJI=";R:LPRINT "SUUJI=";R
520 PRINT "DATASUU N=";N:LPRINT "DATASUU N=";N
530 FOR I=1 TO N
540 PRINT USING "DATA(###)=#####.## ";I;S(I)
550 LPRINT USING "DATA(###)=#####.## ";I;S(I)
560 NEXT I
570 RETURN
580 CLOSE #1:END

```

データの  
キーイン

フロッピー  
への書き込み

ETUKO 9の  
プログラムリスト  
です。

変数の  
クリア

フロッピーから  
の読み出し

135番のPRINTとLPRINTは後に何もありませんが、これはそれぞれ表示と印字を1行空送りするためのものです。

DATA(1,5)=1555.55

NAME(L)=  
NAME(R)=  
SUUJI=0

PUT、GETでは音とか光での反応がないので拍子抜けしました。OPEN文でドライブを起動し、CLOSE文ではこれを停止させるだけなのかと思い、



GET文のすぐ後にCLOSE #1を入れて実行してみました。すると、3番目のプリントが2番目のクリアプリントと同じになりました。CLOSE #1によって、バッファ#1が消えてしまったみたいです。

CLOSE #1

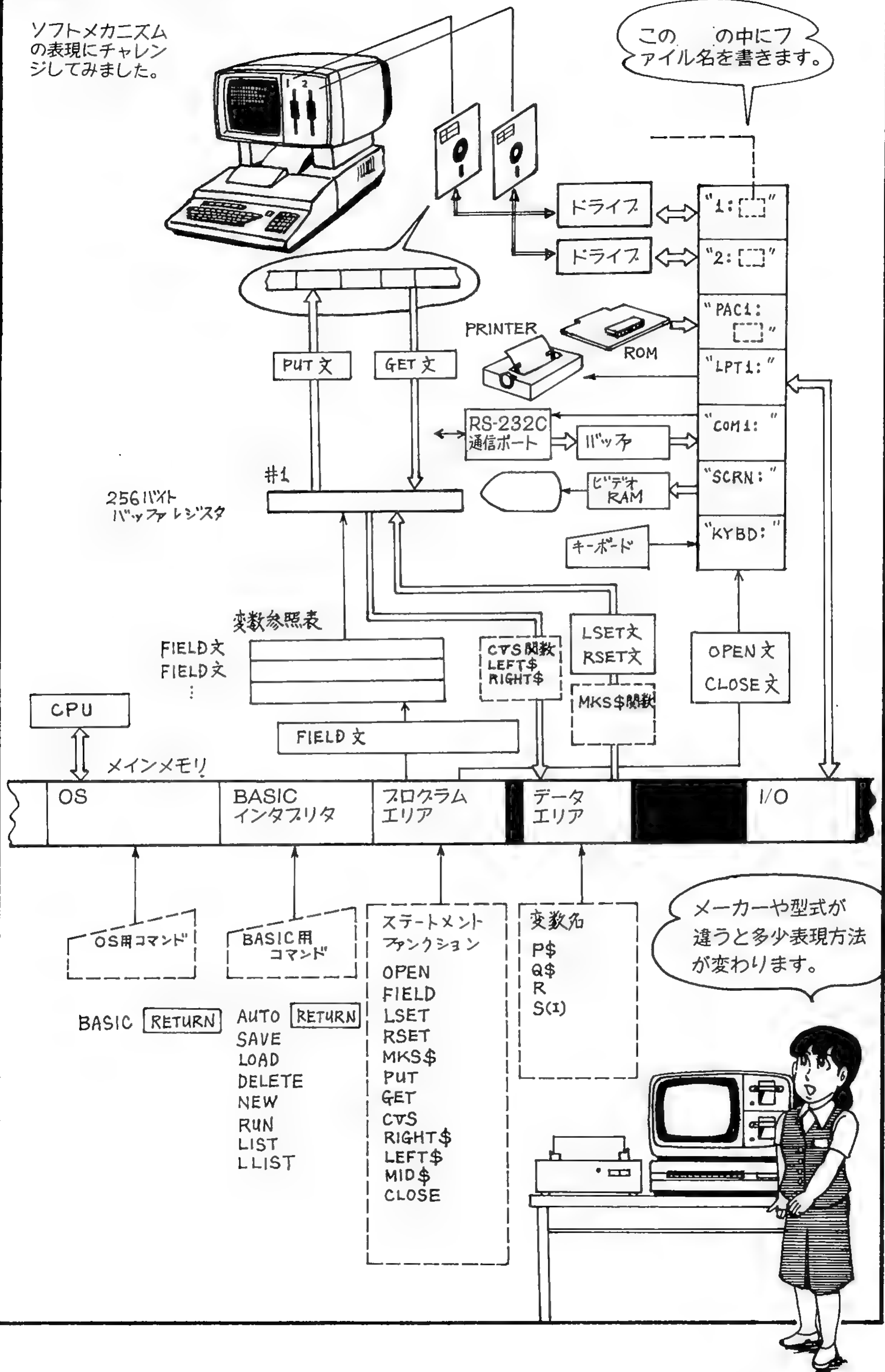
#1

A\$, B\$, C\$, D\$



ソフトメカニズム  
の表現にチャレン  
ジしてみました。

この の中にフ  
ァイル名を書きます。





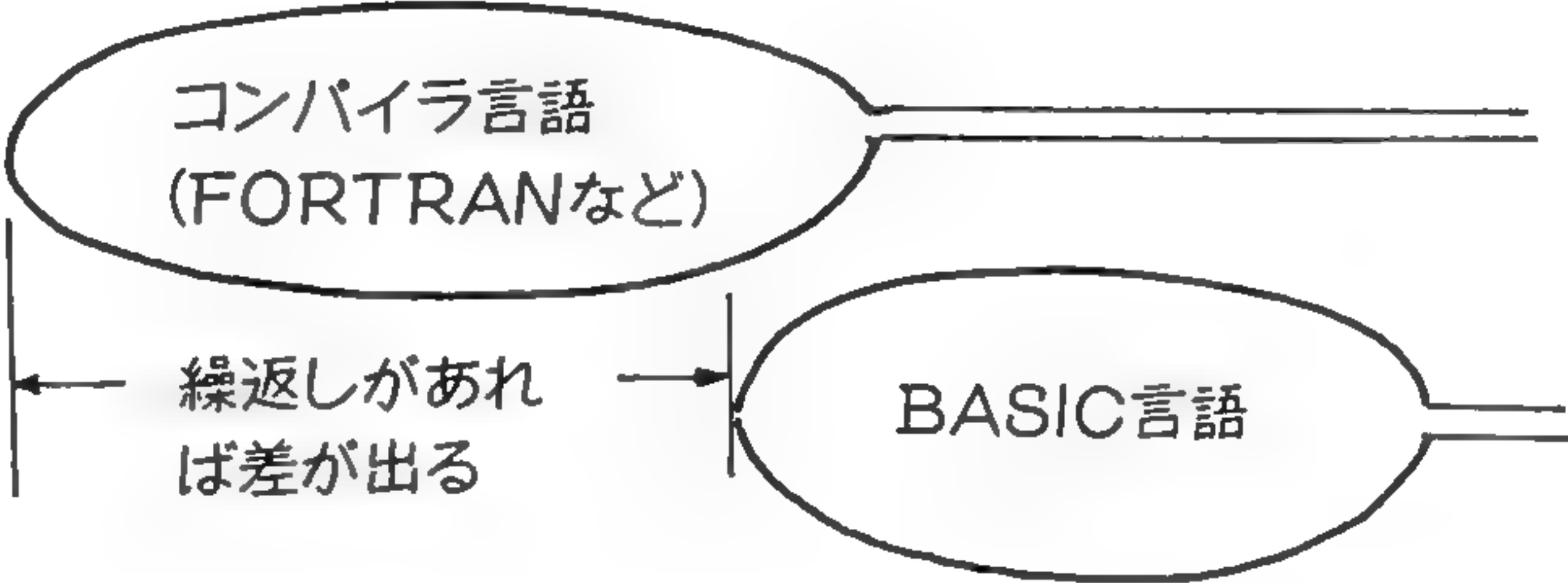


# BASIC Q&A

\* BASICは処理スピードが遅い？

使う処理の対象によっては、確かにコンパイル型言語によるプログラム処理よりも遅く、気になる場合があります。たとえば低速I/O回数の少ないグラフィック処理、それに解析用技術計算などです。

しかし、最近の各メーカーの提供するBASICは、それぞれ工夫を加え、グラフィックスやI/O命令も充実させてきており、十分実務に活用できる環境をつくりあげつつあります。しかし、それを使いこなし、活用する人材の層がまだまだ薄いのが現状です。



\* BASICは互換性がないのでは？

高級言語であるBASICは、どのパソコンでも走らなければならないところですが、現実にはA社のマニュアルで作ったプログラムがそのままB社のパソコンで走るということはありません。しかし、同じBASICなのでからまるっきり違うということはなく、少し修正すれば走るのが普通です。とくにI/O関係の命令がまちまちです。

RS-232Cファイルオープンもこう違う

A社	B社
<pre>OPEN "COM1:4N81" AS 1 (ひとつの文で入出力宣言)</pre>	<pre>OPEN "COM:4N81NN" FOR INPUT AS #1 OPEN "COM:" FOR OUTPUT AS #1 (入力と出力を別々に宣言)</pre>

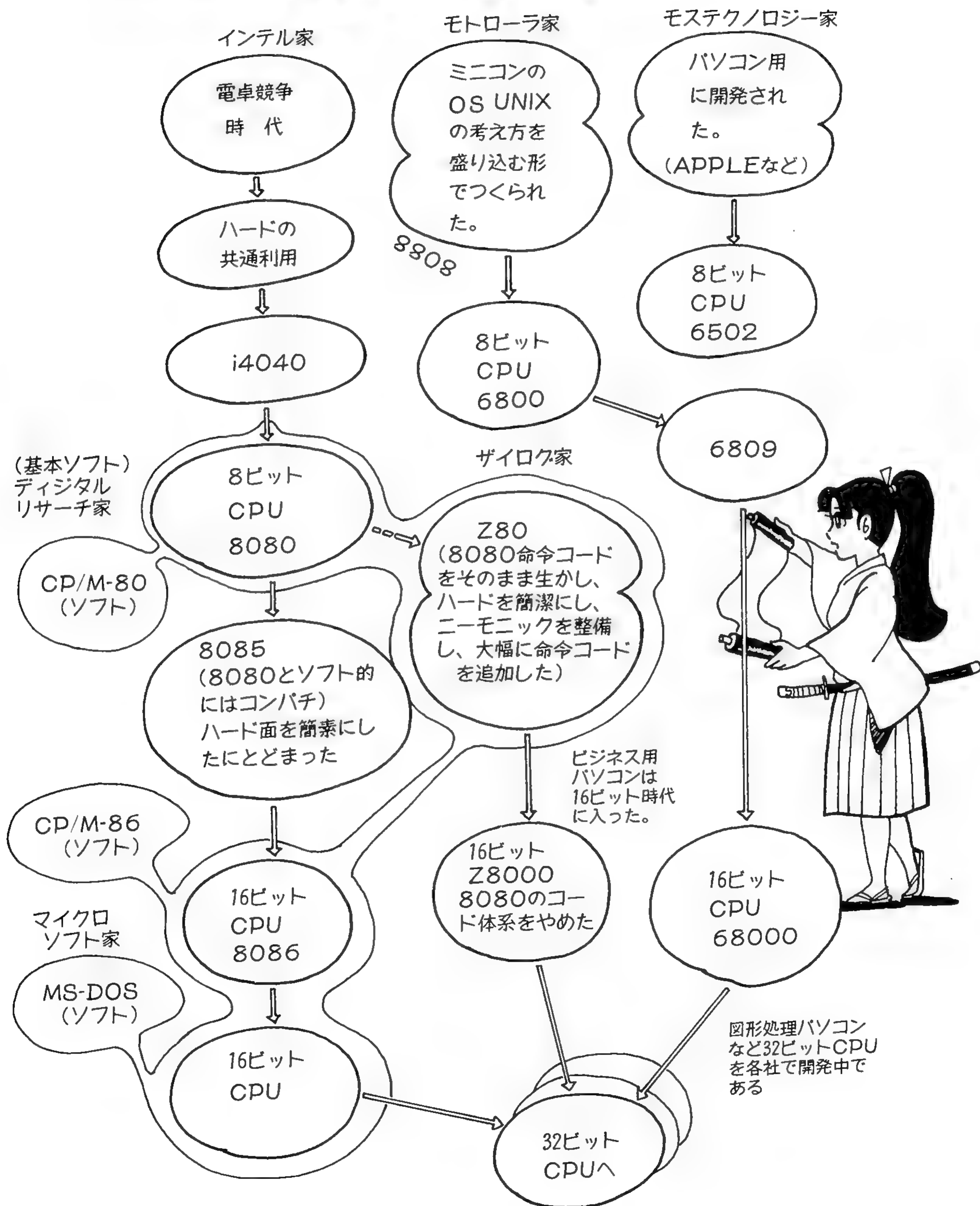
\* PASCALやCが注目されているようですが？

両方共コンパイル型言語です。PASCALは、はっきり言って教授が学生のプログラムを採点しやすいように作ったもの、という印象です。たとえば印字のための命令すらなく、ただCRTに結果を表示できるだけです。それをしようとするればそのマイコンのマシン語が使える能力がなければダメなのです。CはPASCALよりもさらにアセンブラに近い低級言語です。これも標準ではCRTに表示するだけですから、実用的に活用するためには高度の知識が必要です。これらのソフトは10数万で買えますが、ディスク付のマイコンシステムでCP/Mが使える環境が必要なため、誰でも簡単にトライできるというものではありません。



# B マシン語がわかれば もっと使いこなせる

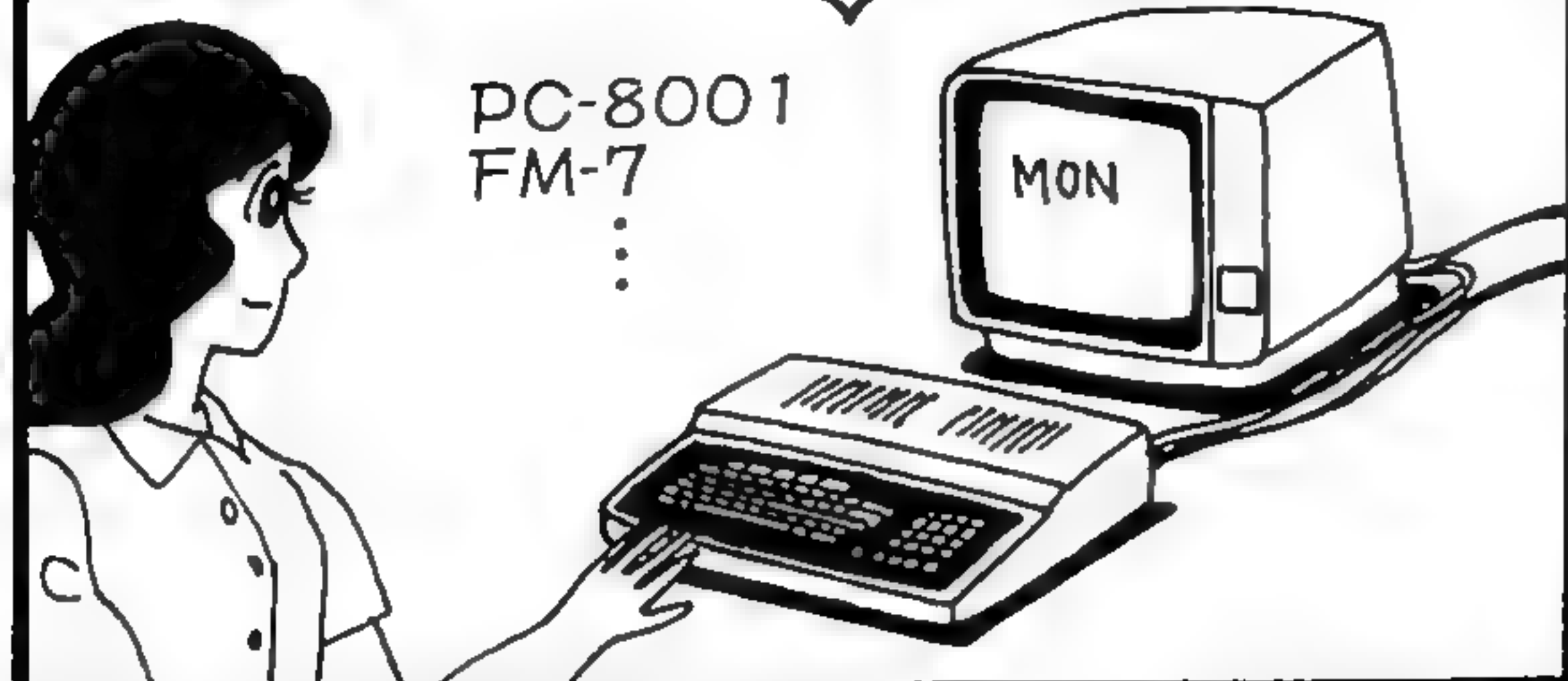
CPUの系図(言葉の違いは育ちの違い)





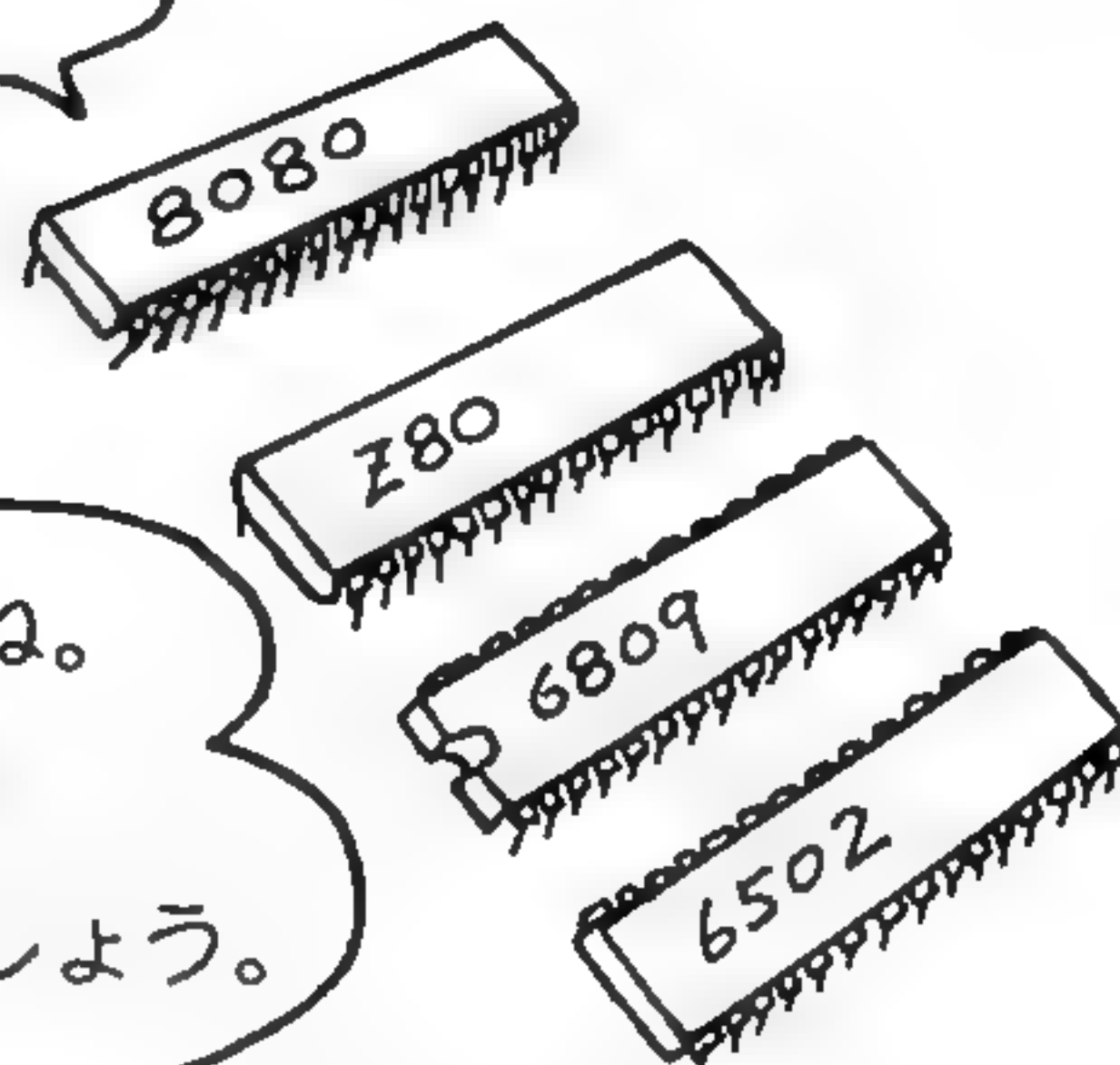
## ①マシン語との出会い

パソコン（ROM型）ではMON コマンドでマシン語らしきものに出会うことができます。



シーン

ね、ね。お話ししましょう。



ま、とにかくMON[RETURN]とKEYをたたいてみましょう。すると\*印が現れました。

NEC PC-8001 BASICver  
Copyright 1979 by Micros

OK ← 電源ONでここまで表示  
MON ←  
\* ← KEYインした内容

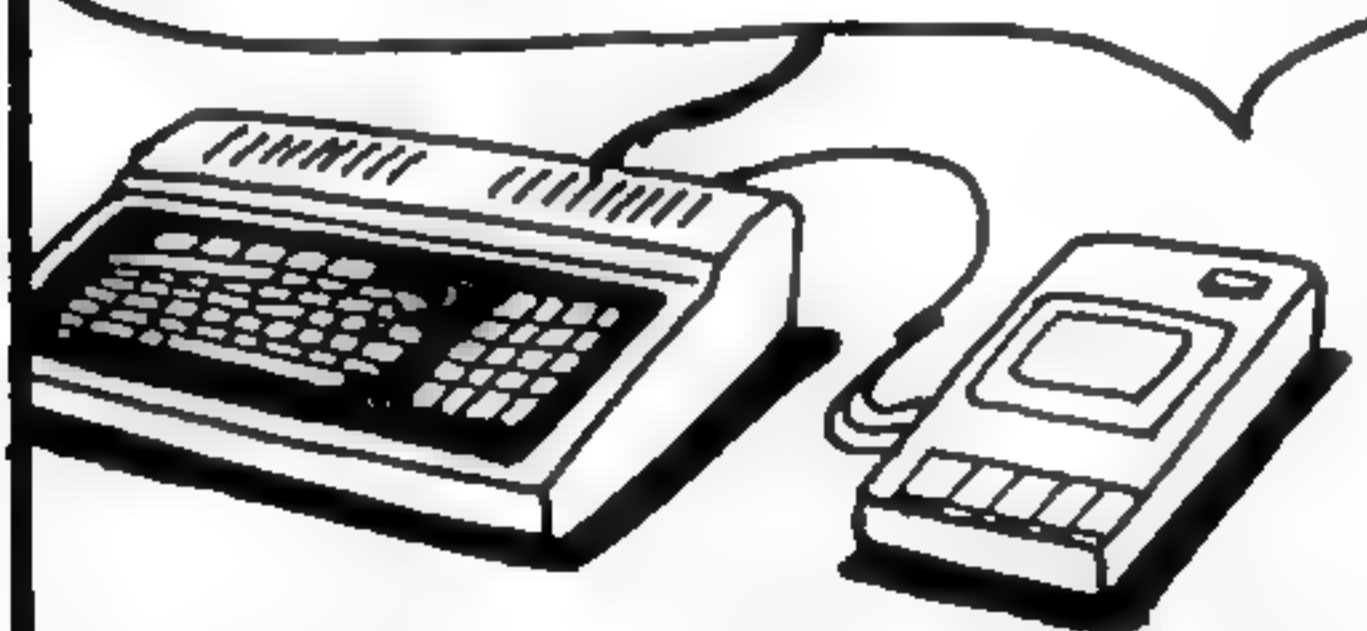
モニタって何？ ということになりますが、それはこれからお話しする、ワンボードマイコンを使って説明します。



このMONは（モンスタ）のモンではなく（モニタ）の略語です。



たとえばインベーターなどのマシン語のテープを読み込ませたい時は[L][RETURN]とします。うまく読み込みが完了したらまた\*表示になります。

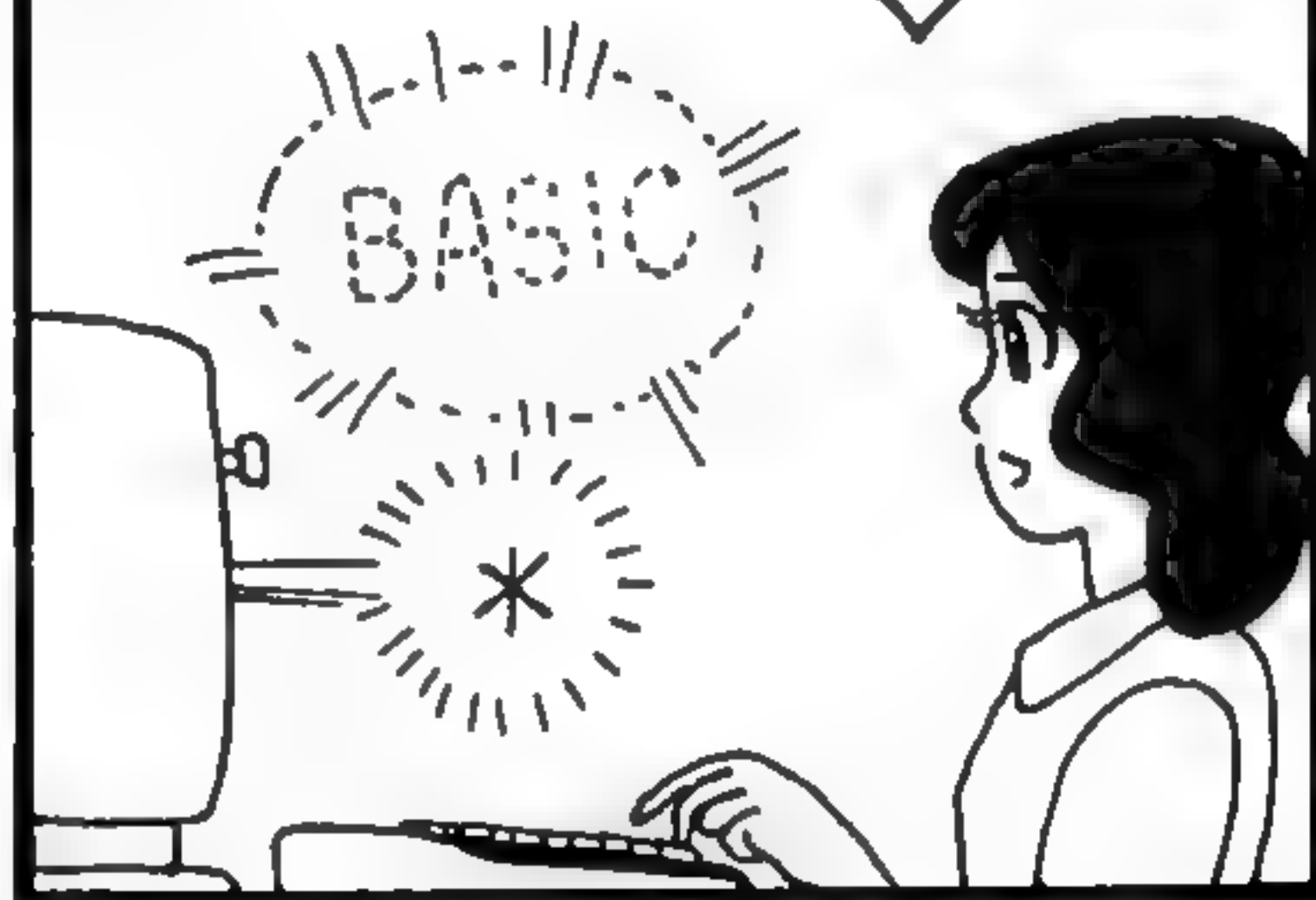


カセットを入れ再生状態にしておく。

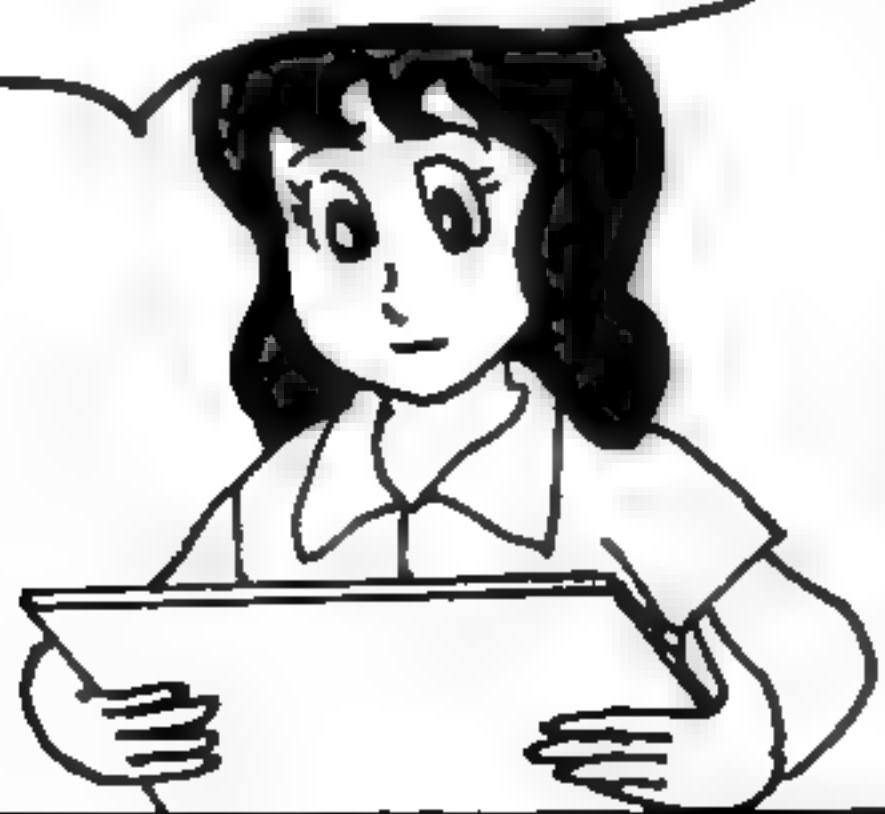
PC-8001のモニタのコマンド

S...RAMにデータを書き込む  
D...メモリの内容を表示する  
L...テープからマシン語を入れる  
LV...ロードベリファイ  
W...テープにマシン語を録音  
G...プログラムの実行  
CTRL B...BASICに戻る  
TM...テストメモリ  
(FM-7/8は別のコマンド)

これが、モニタのコマンド待ちの状態です。BASIC言語は使えなくなりました。



ここで表示されたC3とか06とかいうのが実はマシン語（に近い）言語なのです。まずワンボードマイコンでお話ししましょう。



DC000,COOF[RETURN]  
とキーインするとC000番地からCOOF番地までのデータが表示されます。

\* DC000, COOF  
C000 C3065839...  
C007 003832FC...  
\*

↑ アドレス      ↑ 数字2つがひとつのデータ

プログラムの実行はG（ゴー）コマンドです。テープの解説に書いてあるアドレスを入れるとプログラムが走ります。  
GC000[RETURN]





## ②ワンボードマイコンとモニタ

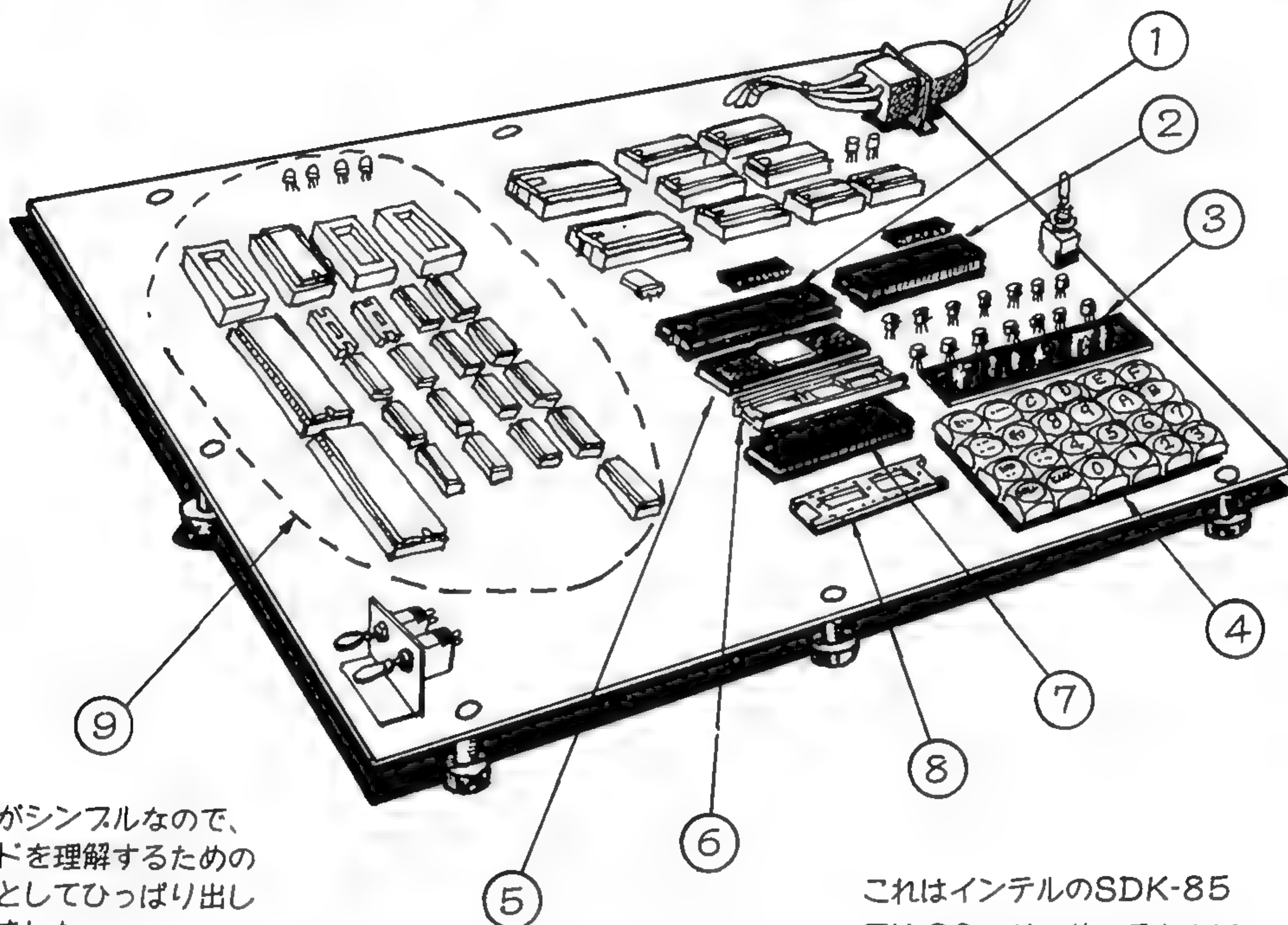
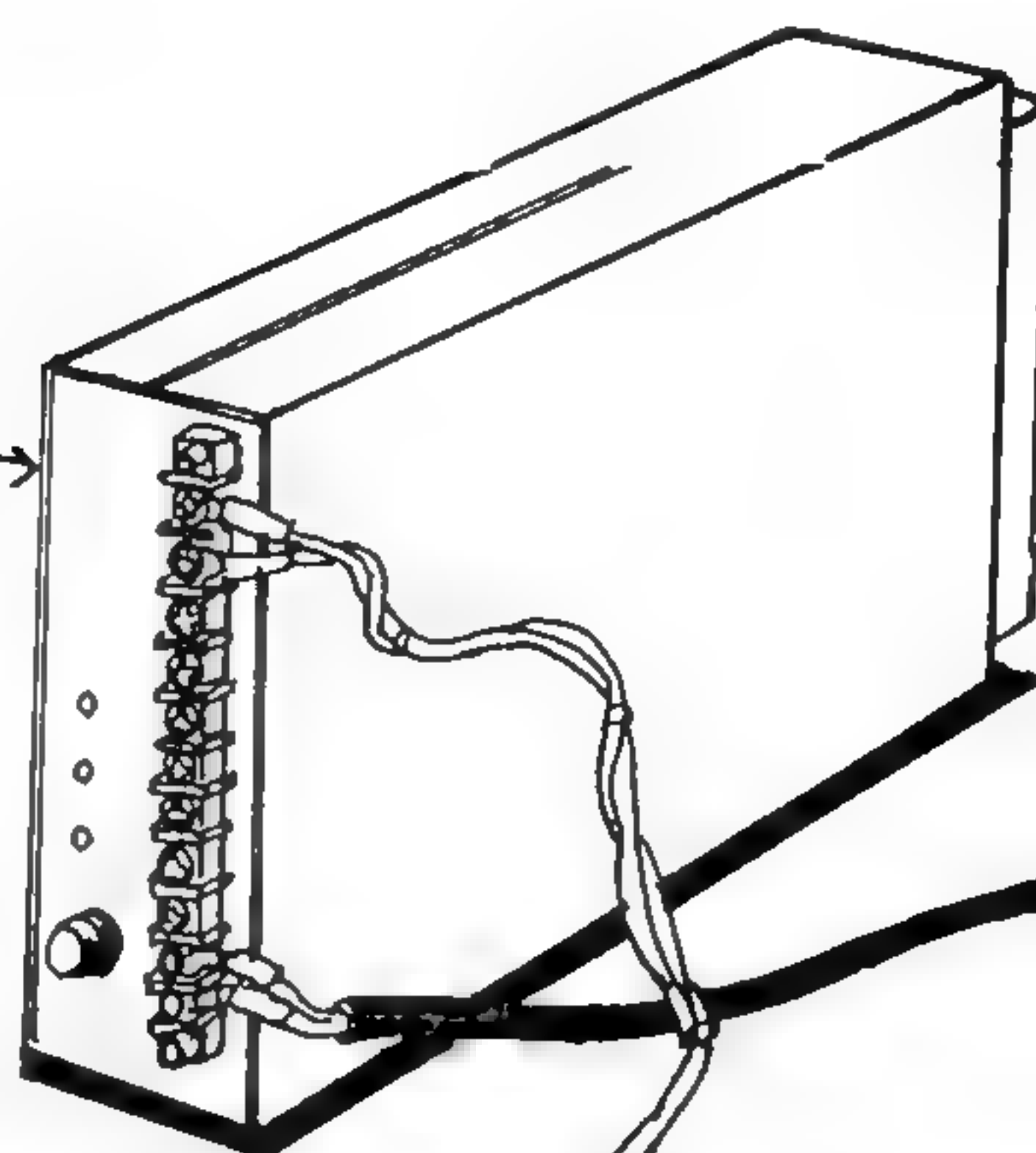


**HELP**

CRT なんてないもんね。7セグメントのLED表示

その昔、一世を風靡したワンボードマイコンです。

DC5V AVR→  
直流DC 5V  
の定電圧装置  
ICの駆動用  
です。



構成がシンプルなので、ハードを理解するための教材としてひっぱり出してきました。

これはインテルのSDK-85 TK-80、その他いろんなワンボードマイコンがありました。

- ① CPU8085 (8080とほとんど同じ)
- ② キーボード・ディスプレイコントローラ8279
- ③ 7セグメント表示器 (アドレス4桁データ2桁)
- ④ 16進キーボード (プログラムは0~9、A~Fのキーを使ってここから入れます)。
- ⑤ ROMメモリ
- ⑥ ROMメモリ増設用ソケット
- ⑦ RAMメモリ
- ⑧ RAMメモリ増設用ソケット
- ⑨ オプション用ユニバーサル部分

今は7と表示している。



7セグメント  
LED

0~9を入れて128通りの表示のしかたがあります。



はじめての人には、このヘキサという数字にも戸惑われるかもしれませんが、わからなくても先に進んでください。そのうちわかります。



HEXA

0、1、2、3  
4、5、6、7  
8、9、A、B  
C、D、E、F

このキットにはすでに固定プログラムが入っていますので、私たちは16進数(ヘキサ)でデータを扱うことができます。

モニタ  
プログラム



これが、このキットのインプット部です。プログラム、データは、このキーを使ってインプットします。

RESET	INTR	C	D	E	F
SINGLE STEP	GO	8 H	9 L	A	B
SUBST MEM	EXAM REQ	4 SPH	5 SPL	6 PCH	7 PCL
NEXT	EXEC	0	1	2	3

プログラムの内容はアドレス表示用4桁に 'HELP' を点滅させるものです。



この表示器を使うプログラムがマニュアルにありますので、これをインプットして走らせてみましょう。



アウトプットは、この6桁のLED(発光ダイオード)表示です。データを扱う時は、アドレス4桁データ2桁で表示されます。

アドレス用

データ用

8.8.8.8. 8.8.

'HELP' の点滅プログラム  
(SDK-85以外のキットには使えません)

(右のリストのアセンブラプログラムですが、このキットでは使えません。)

アセンブラ ニーモニック  
LXI SP, 20C8H  
MVI A, 1  
MVI B, 0  
LXI H, 2004H  
CALL OUTPT  
DPY: MVI A, 0  
MVI B, 0  
LXI H, 2000H  
CALL OUTPT  
LXI D, OFFFFH  
CALL DELAY  
MOV A, 0  
MOV B, 0  
LXI H, 2004H  
CALL OUTPT  
LXI D, OFFFFH  
CALL DELAY  
JMP DPY

アドレス	データ
2000	10 0E 11 12 15 15 15 15
アドレス	プログラムデータ
2010	31 C8 20
2013	3E 01
2015	06 00
2017	21 04 20
201A	CD B7 02
201D	3E 00
201F	06 00
2021	21 00 20
2024	CD B7 02
2027	11 FF FF
202A	CD F1 05
202D	3E 00
202F	06 00
2031	21 04 20
2034	CD B7 02
2037	11 FF FF
203A	CD F1 05
203D	C3 1D 20

この簡単な例でマシン語とCPUの働きを理解してもらおうと思います。

この部分がパソコンのMONモードで使ったマシン語にあたる場所です。このプログラムは2010番地からスタートします。

マシン語





この例題のプログラムは、アドレス2010 からスタートするようになっています。

アドレス(番地)データ

2010	31 C8 20
2013	3E 01



RESET キーを押すと、-8085 を表示します。次に SUBST MEM キーを押すと、小数点がひとつ点灯され、あとは消えます。

- 8 0 8 5

  
 ↓  

.

ではプログラムしてみましょう。電源をON(オン)にすると、6桁全部フルコードの目となります。どうしてこうなるかという、このモニタプログラムがこうなっているからです。

8.8.8.8. 8.8.

次に、新たにプログラムしたいデータ 3 1 NEXT と押すと、新たなデータにおきかえアドレスは次へ2011となります。

2 0 1 0 3 1

残っているデータ

2 0 1 1 4 6

方法は、キーを 2 0 1 0 と押し、NEXT と押すことで、2010番地とそのデータが表示されます。

2 0 1 0

残っているデータ

2 0 1 0 C 5

ちょっとこのおかしい書き方について説明しておきましょう。これは、2010-31 2011-C8 2012-20 と書くところを略して 2010 31 C8 20 としているのです。どうしてこんな書き方をするのかということはもう少しあとで説明します。数字(?)2桁でひとつの記憶単位になってます。

これだけのプログラムを入れるのに、何回キーを押したと思いますか? 149回です。ああ、しんど。



あとは、同じことのくり返しです。順々に書き込んで 2 0 NEXT でプログラムの書き込み完了です。

203D C3 1D 20  
 ↓  

2 0 3 F 2 0

2進  
0011 0001  
↑ 16進  
(( 31 ))  
同じなのよ。



このあとで書き込んだ8桁分のデータは文字として表示させるためのものです。

アドレス データ

2000	10	... H
2001	0E	... E
2002	11	... L
2003	12	... P
2004	15	... ブランク
2005	15	... "
2006	15	... "
2007	15	... "


交互に表示させると  
HELP  
が点滅することになる

RESET SUBST MEM 2 0 0 0 NEXT 1 0 NEXT と順に押して、アドレス2007までのデータを書き込みます。

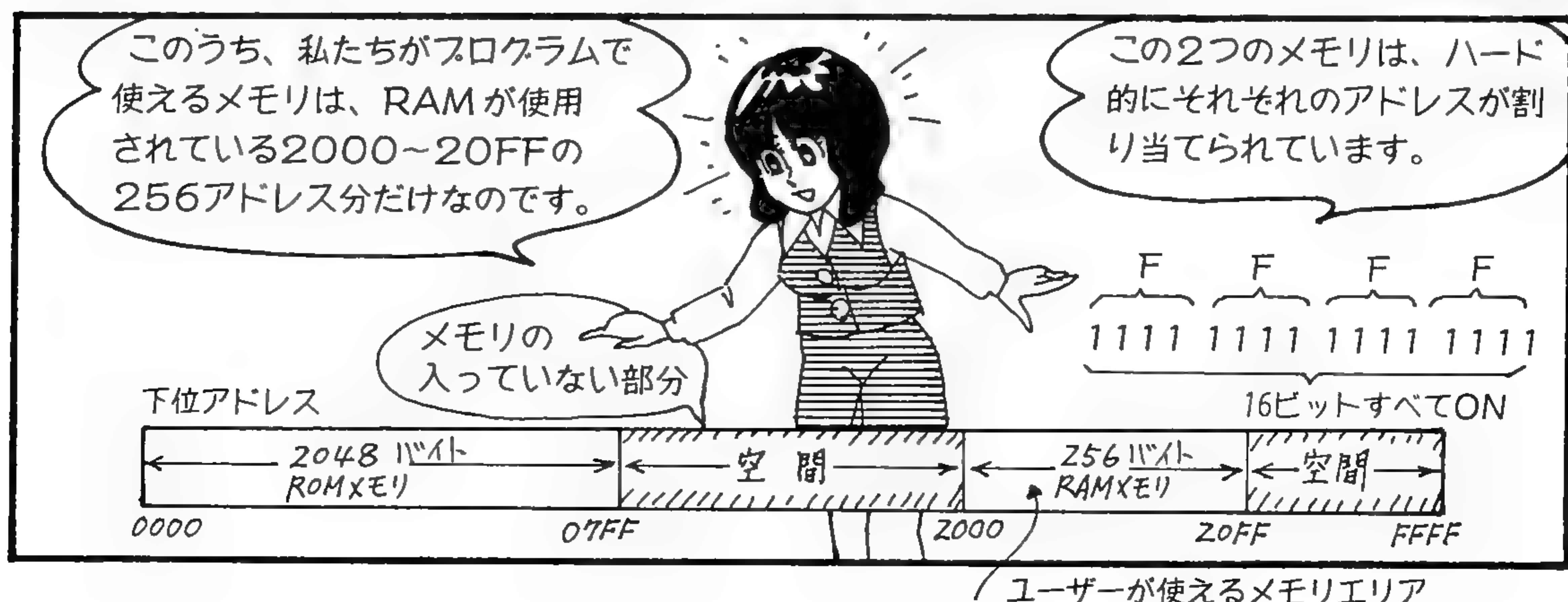
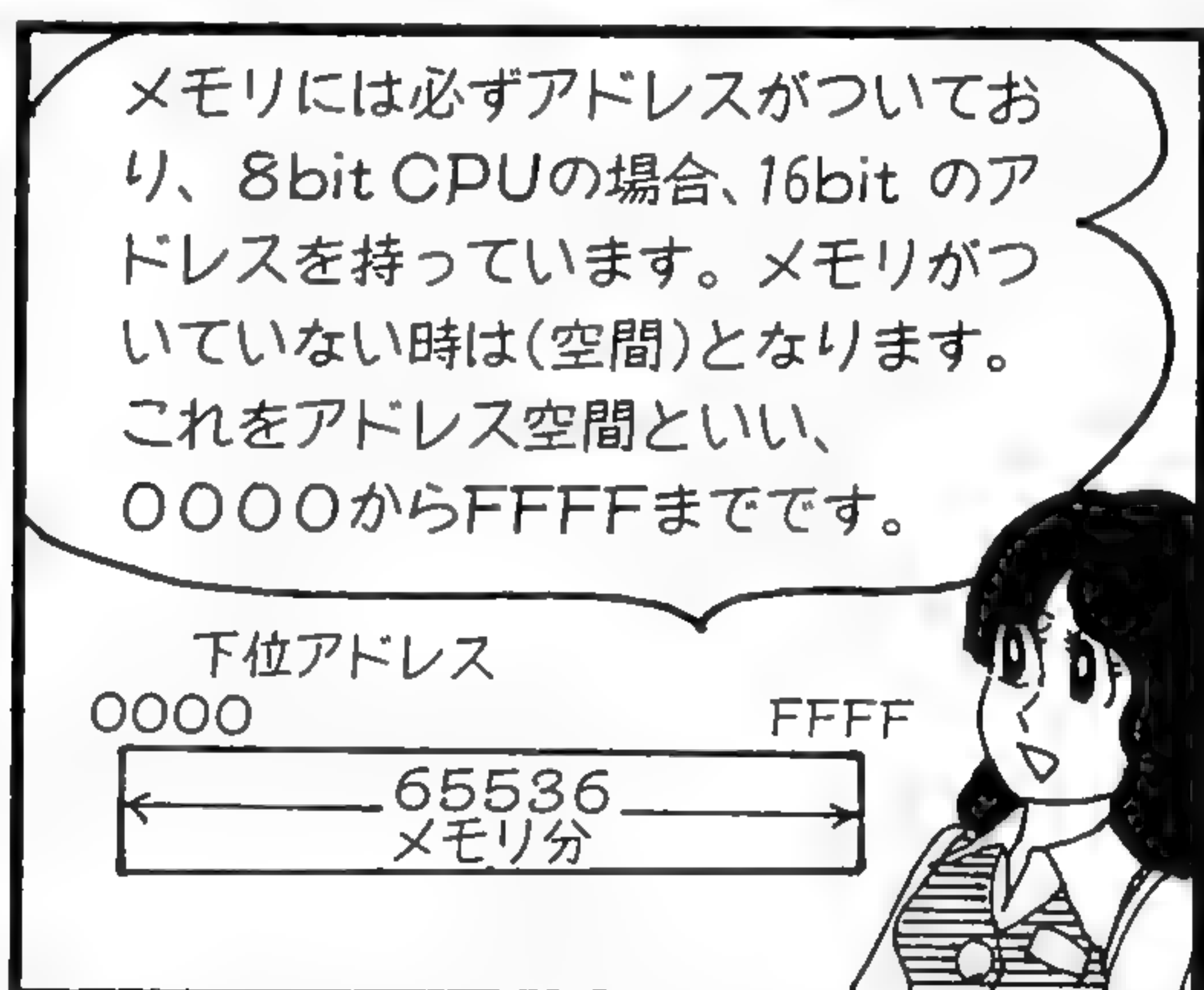
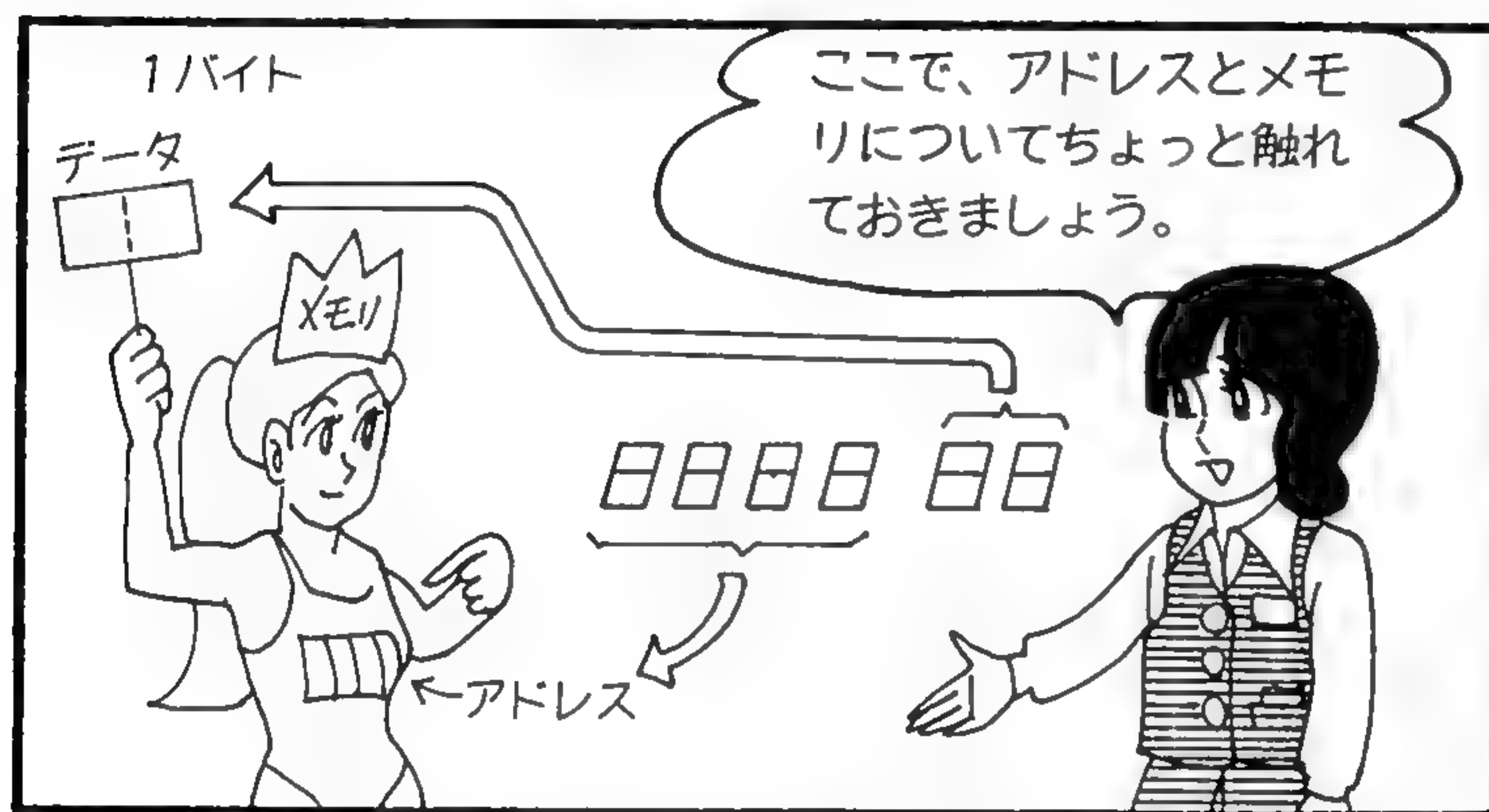
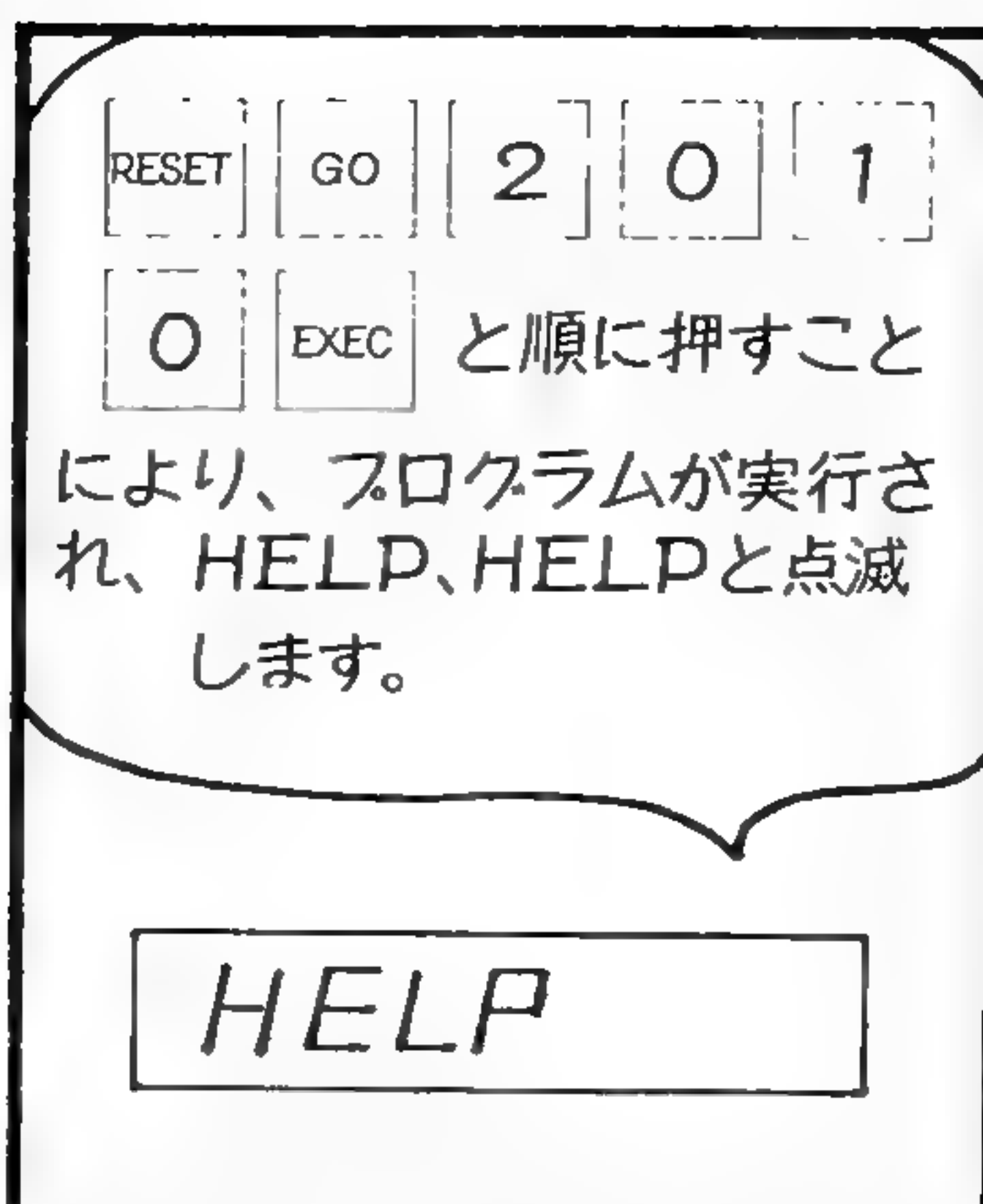
2 0 0 0 1 0

でも、この例題では、さらにプログラムで使うデータも書き込めと書いてあります。

2000 10 0E 11 12 15 15 15 15









電源をONにすると、CPUはアドレス0000からスタートします。ここに、モニタプログラムが入っているのです。

電源 ON

ROM 0000

モニタプログラム

07FF

表示

私たちが使えないメモリなんて意味がないと思うのですが、実はこれにはすでにプログラムが入っているのです。

プログラム済

ROMメモリ

もし、このモニタがなかったら、つまり最初から自分で全部やろうと思ったら、

私たちが、16進数を使ってプログラムできるのも、このモニタプログラムのおかげです。

16進キーコマンドキー受け付けます

モニタ

ROM

電源をONした時に、LEDが全点灯したのは、このモニタプログラムが実行されたからなのです。

表示

この時のデータ表現は2進数、とてもやってられません!!

0010 0001

0100 1100

お手上げ

アドレス16ビット分のスイッチ16個を、アドレスバスにつなぎ、データ8ビット分のスイッチ8個をデータバスにつないで、ひとつひとつプログラムデータをメモリに書き込まなくてはなりません。

アドレス16ビット

データ8ビット

WRITE SWITCH

書き込みスイッチ ON

アドレスをセットして...

データをセットして...

パソコンのモニタモードは、こうしたワンボードマイコンユーザーにとってはまさに福音だったのです。パソコンにはCRTがついているし、はるかに使いやすくなったのです。

モニタプログラム

MON

そして、もう一度電源をONした時、RAMはデータラメな内容になっているので、もう一度プログラムしなおさないと、HELPプログラムも実行できません。

RAMメモリ

データ

HELPプログラム

2000

20FF

一方、RAMメモリの方は、電源をOFFにした時、その記憶内容は消えてしまいます。

RAM

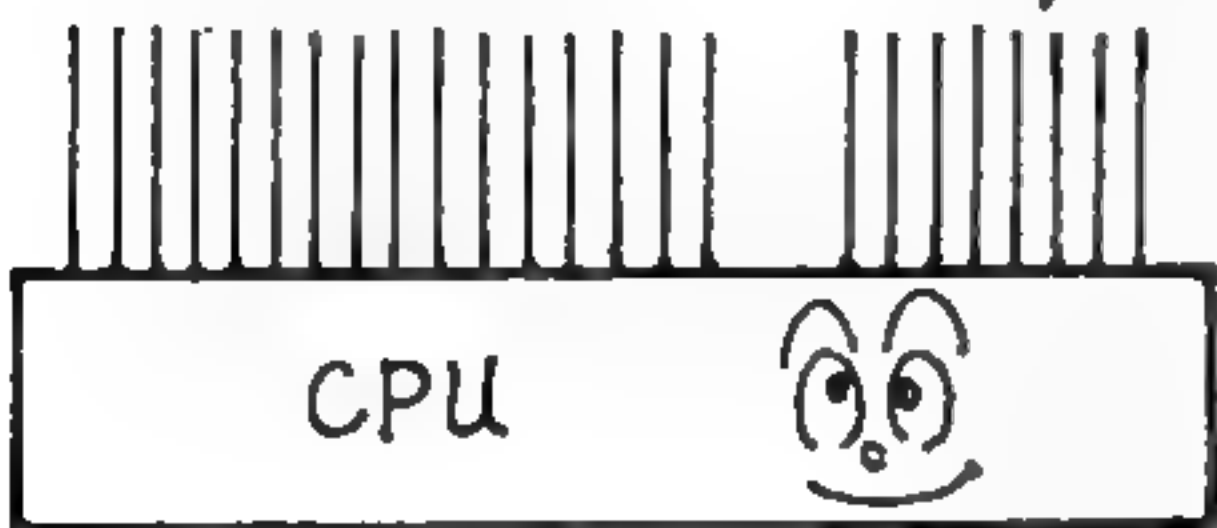
忘れる



コンピュータが使うのは、2進数です。このことは大型、小型に関係ありません。

アドレス16bit

データ8bit



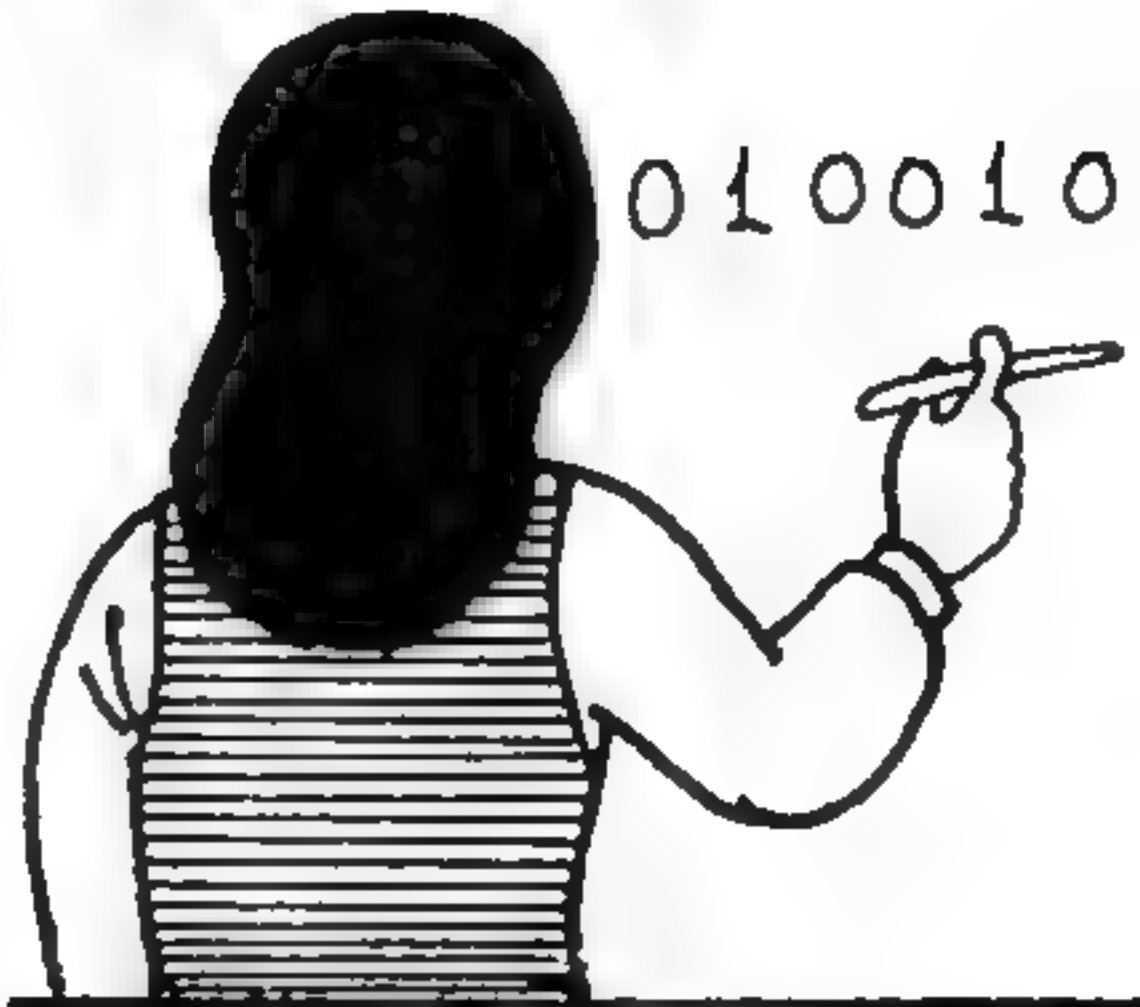
私たちが日常使っているのは10進数ですね。2進数も16進数もなじみのない数字です。

1000円  
10m  
100点  
100kg

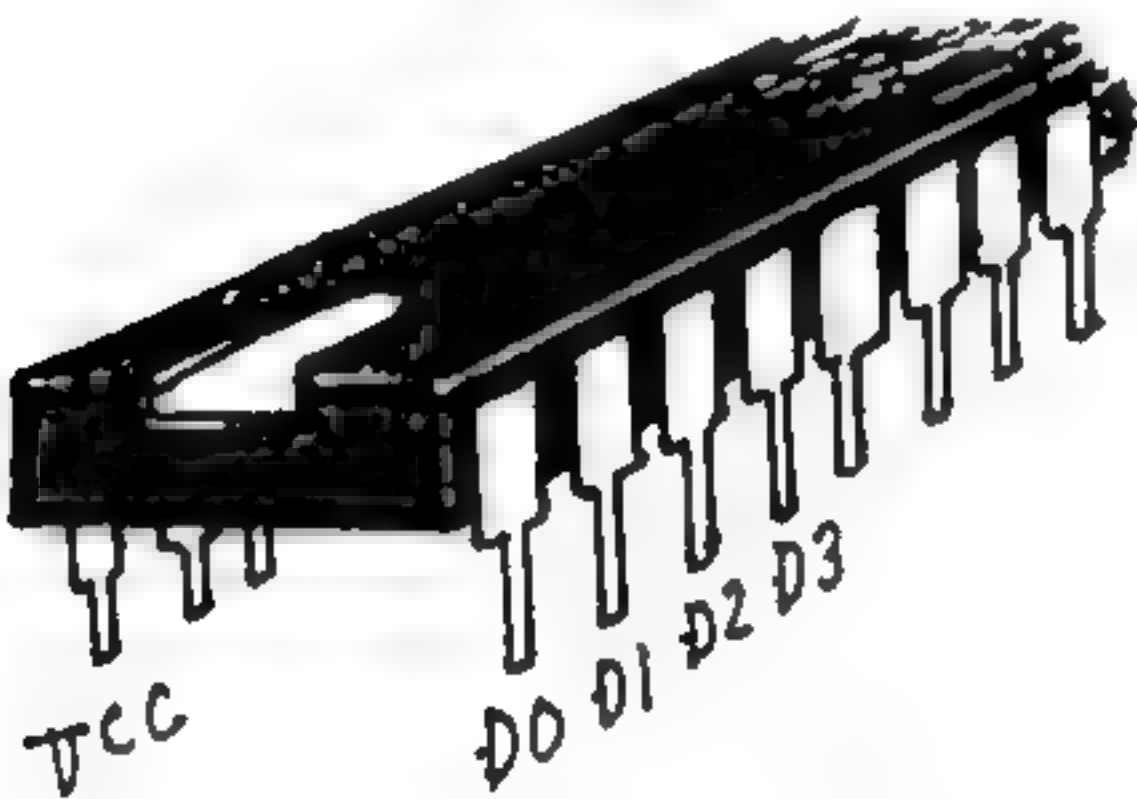


### ③マシン語の『いろは』は 0(ロー)か1(ハイ)

0100100



ICの信号ピンには番号がついていて、各ビットの桁がわかるようになっています。



ふつう、bit は0か1で表現しますが、ICなどのロジック回路では電圧信号として、HとかLとかで表現することもあります。

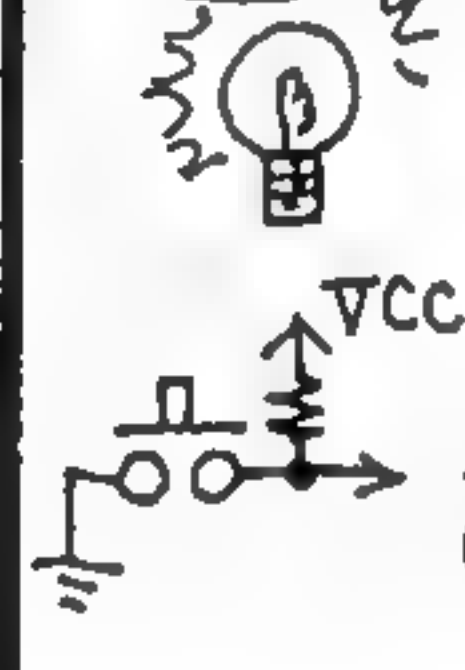


入力		出力
A	B	C
L	L	H
H	L	H
L	H	H
H	H	L

2進数の最小単位が1ビット(bit)と考えていいと思います。

ビット  
bit

電球の点滅



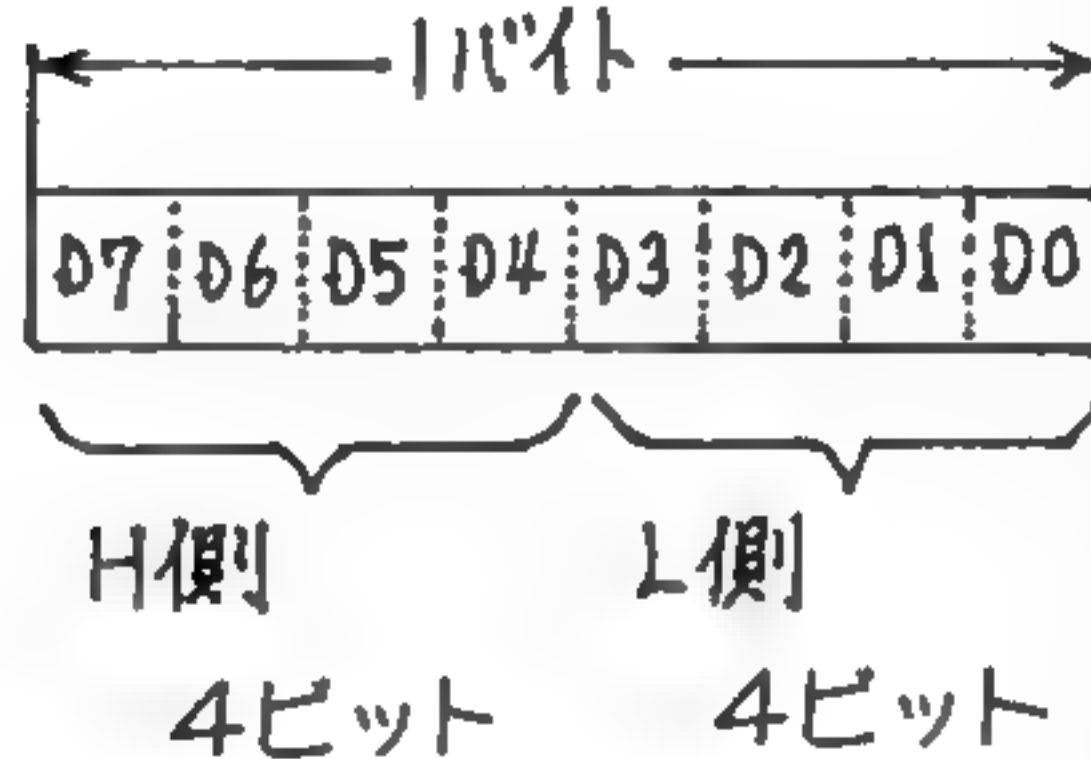
スイッチのON-OFFによる電圧レベルの高低



4ビットでは、1と0で16通りの区別ができます。

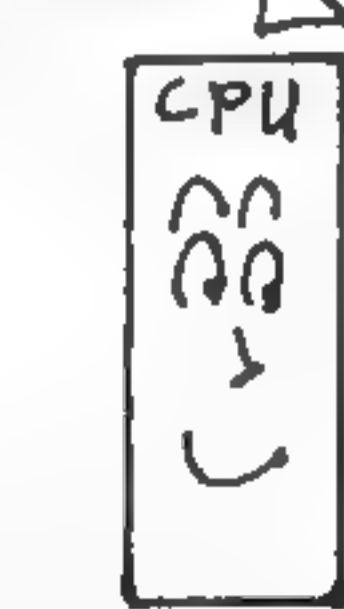
	D3	D2	D1	D0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
C	1	1	0	0
D	1	1	0	1
E	1	1	1	0
F	1	1	1	1

マイコンで使われる機械語は、1バイトの半分の4ビットで区切ったものを使っています。



8bit CPUの場合、メモリも1アドレス8ビット単位が便利です。8ビット単位の情報を1バイト(byte)といいます。

データ11バイト  
8bit



アドレス16bit



CPUは2進表現以外受け付けませんので、人間の方で16進数を使えるプログラムをつくるしかありません。

00110001

モニタプログラム

31



8ビットによる2進表現よりも、16進2桁表現の方が私たちには見やすく、プログラムもしやすいのですが……。

2進8ビット表現

16進2桁表現

0011 0001	31
1100 1000	C8
0010 0000	20

00110001  
11001000  
00100000

31  
C8  
20

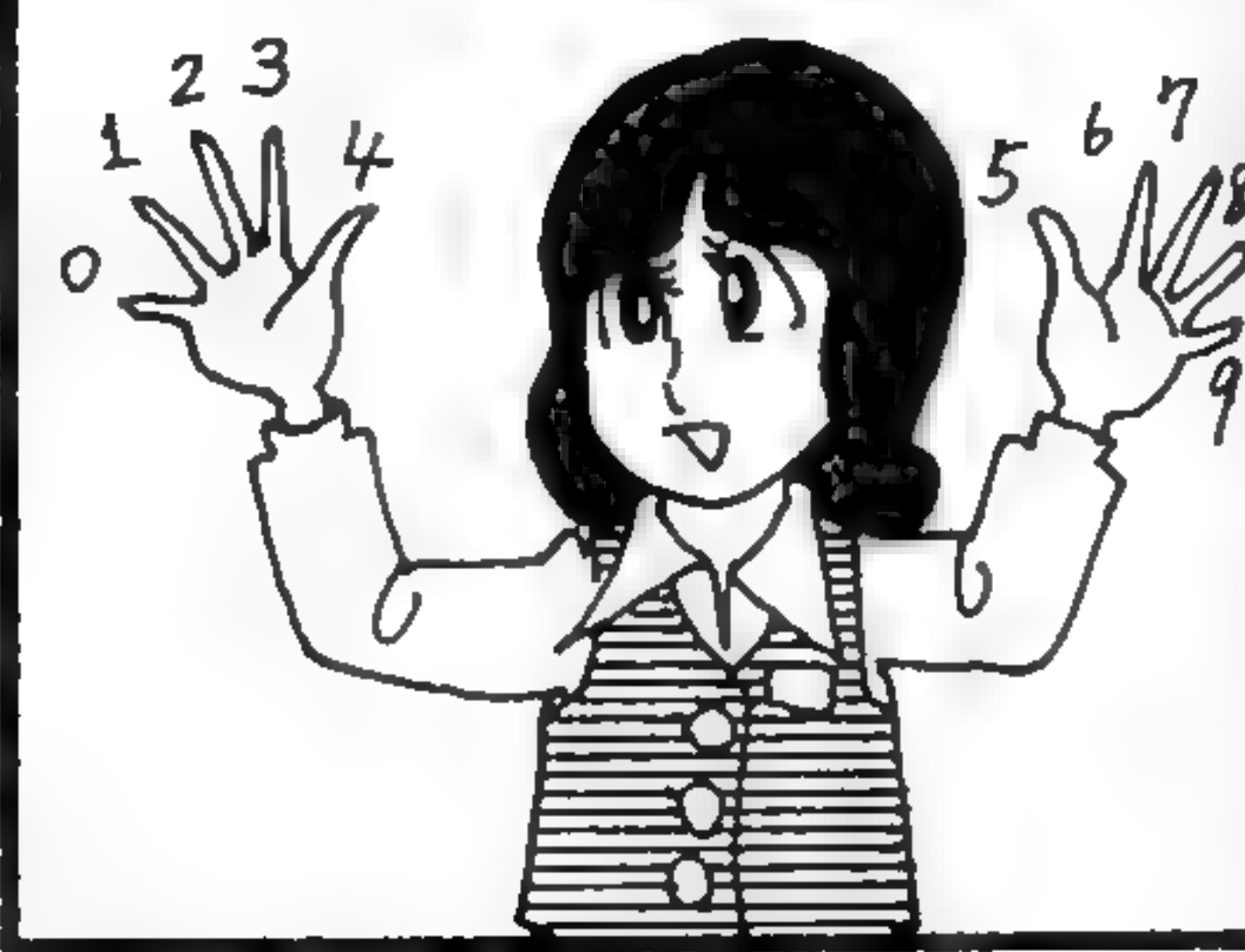




そして16進数については、0～9まではそのまま、A、B、C、D、E、Fの記号を追加しました。



おなじみの10進数については、0～9までの10個の記号の組合せ。



2進数の場合は、0か1のみの組合せによる表現となり、

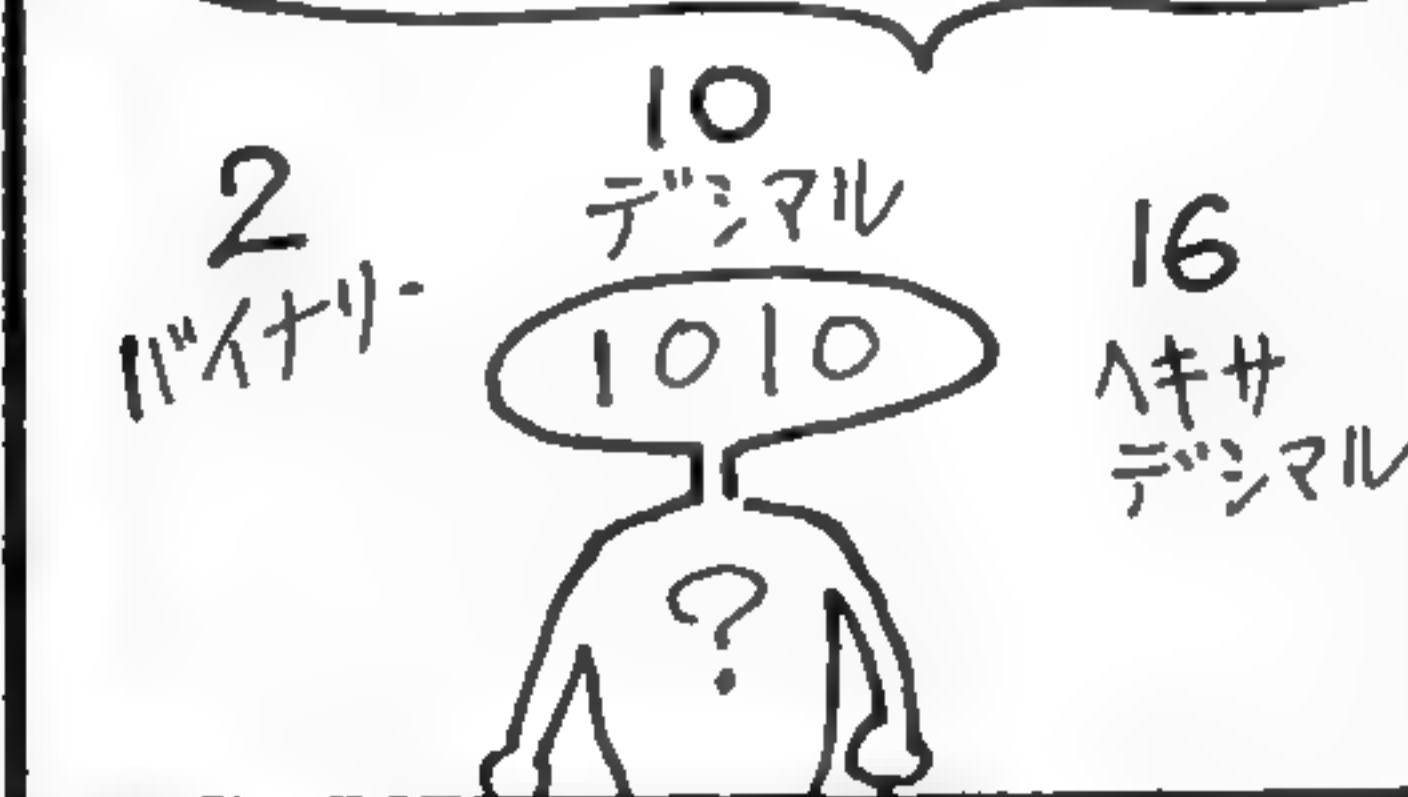


そこで、数字の後にB、D、Hをつけて区別するようにしています。

2進数……1010B  
10進数……1010D  
16進数……1010H



さて、マイコンの世界のように、2進、10進、16進が混在していると、たとえば1010という数字は何に属したのかわかりません。



これら3つにはそれぞれ独立した記号を使えばよかったのにとおもいますが…。

意味が違うのに同じ記号を数字として使うからまぎらわしいのよ。



あとででてくるがアセンブラを使う時にこの考えが必要。

1010B =  $(2^1 + 2^3)D = 10D$   
1010D これは10進だからそのまま  
1010H =  $(1 \times 16^1 + 1 \times 16^3)D = 4112D$

どうです？ それぞれ10進数の10、1010、4112となりました。

ためしに全部10進数に直してみました。



とくに16進の場合で、最初の1文字がA、B、C、D、E、Fのいずれかの場合は、その頭に0(ゼロ)をつけて英文字と区別するようにしています。

ゼロ  
↓  
0A5H

とくにアセンブラ言語として使う時



練習すれば誰でもそれなりに上達します。

4つずつ区切るのがコツ

1101 0111



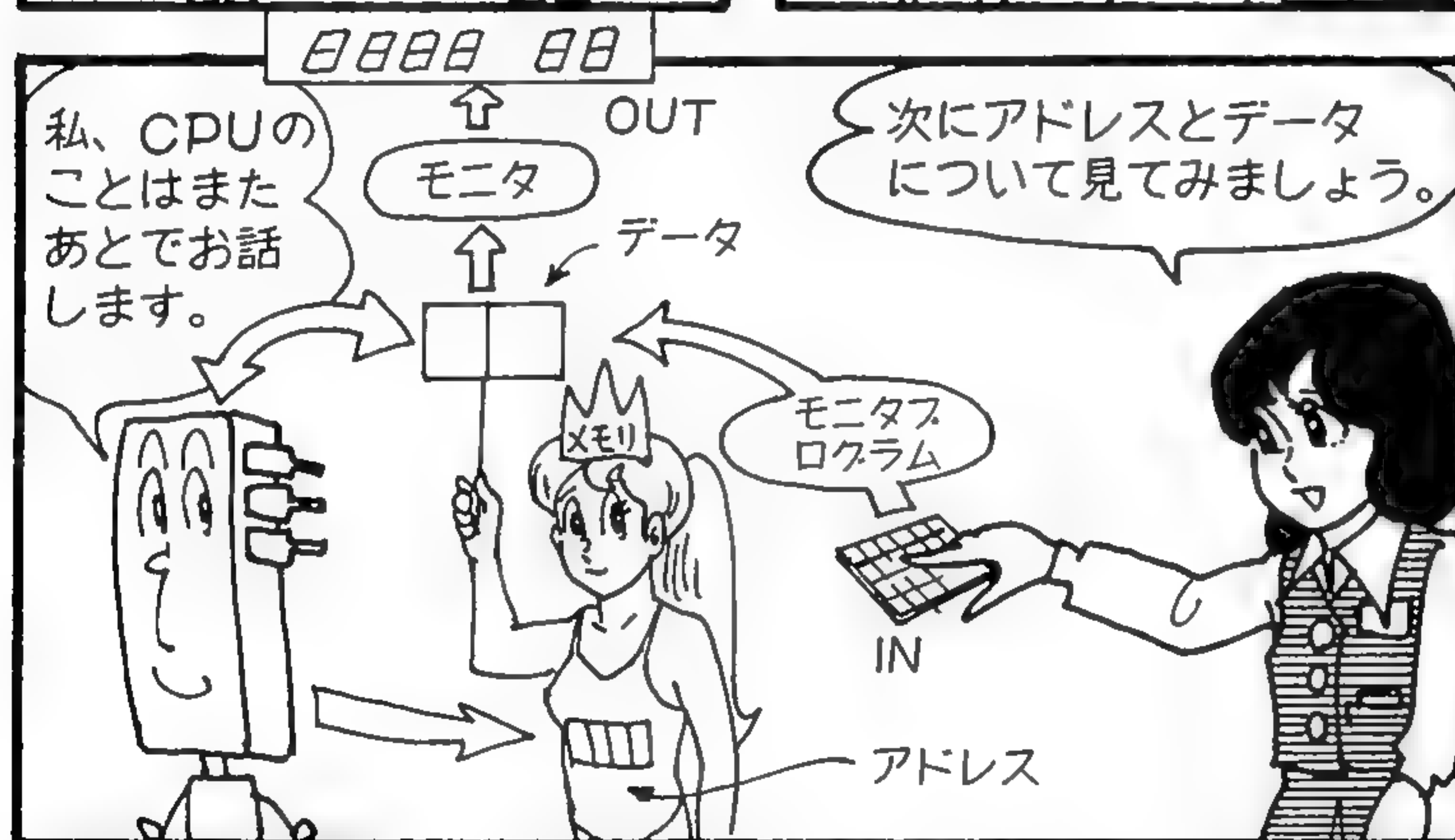
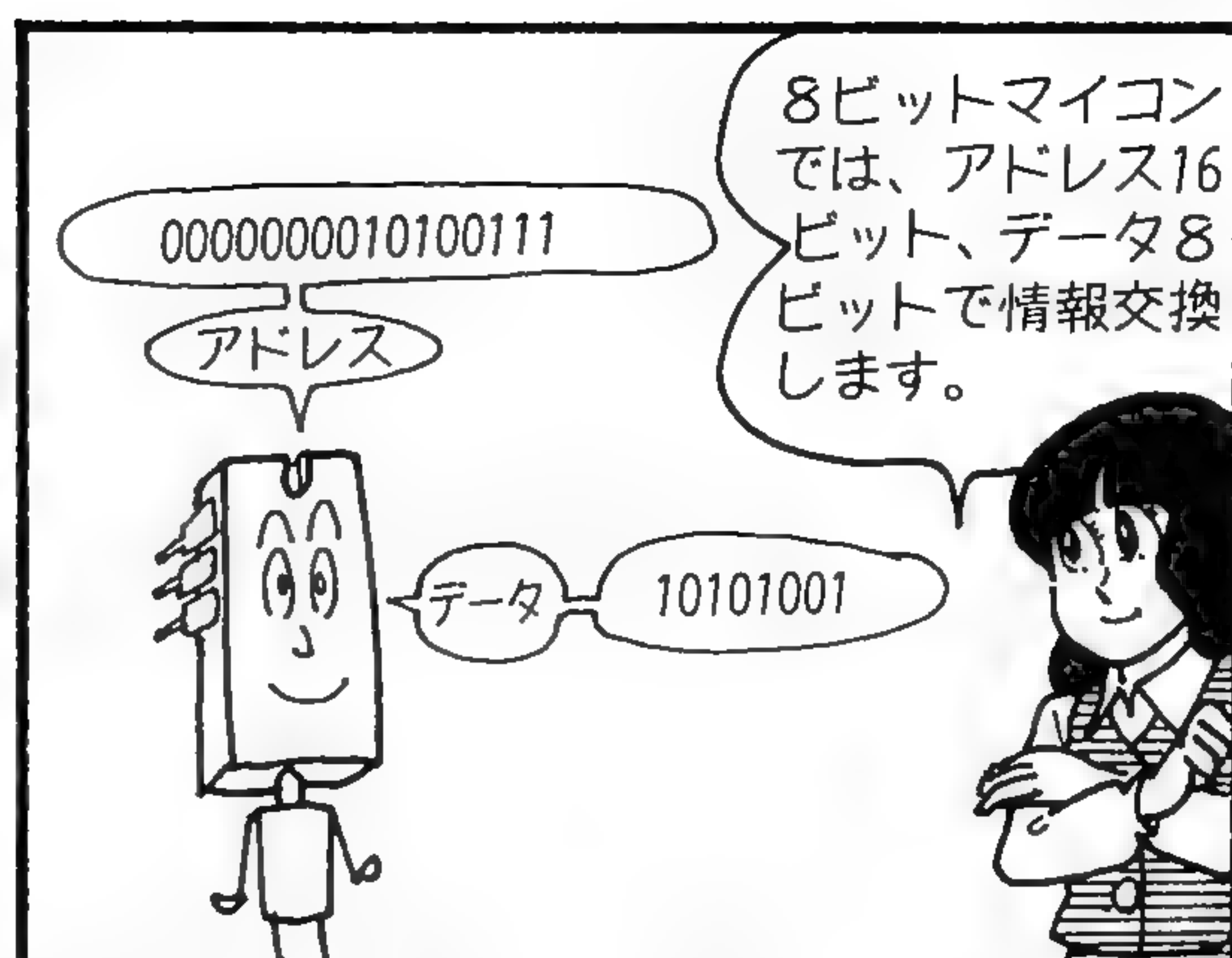
このうちマイコンの世界で最もひんぱんに使われるのが2進と16進の相互変換です。

バイナリ 2進

ヘキサデシマル 16進









そうそう。メモリにはROMとRAMがありましたね。ここでもう少しお話しておきましょう。少しずつマシン語に近づいてきました。

このデータとアドレスとはこれから長いつきあいになります。

実装されたあとは、読出し専用の一方通行で使います。書き込み命令には使えません。

ROMちゃんには、あらかじめデータを書き込んでおきます。

01011101

5

ここで、ROMちゃん、RAMちゃんをご紹介します。

1バイト

たとえばパソコンでMONモードにしておいて0000~5FFF間でデータを書き込んでみてください。

OK  
MON  
\*S 4000  
4000 4C

一方、RAMちゃんは、ユーザープログラムのデータエリアとして、また定数データエリアとして読み、書きができます。

双方向データ

2010 31C820  
3E01

ユーザー

これはPC-8001の場合です。FM-7やその他のパソコンではROM、RAMエリアは違っているはずです。マニュアルのメモリマップを見てください。

これに対して C000~FFFF番地はRAMなのでデータを書き込めばメモリの内容が変わります。

\*D C000  
C000 EF 32 3E 4E  
\*S C000  
C000 EF-01 32-02  
\*D C000  
C000 01 02 3E 4E  
変わっている

Dコマンドでメモリ内容を見るとデータはもとのままです。CPUは書き込み手続きをしたのですがメモリが受け付けなかったのです。ROMメモリだからです。

OK  
MON  
\*S 4000  
4000 4C-01 C1-02  
\*D 4000  
4000 4C 01 C3 75 3F  
変わっていない



各アドレスのメモリには、このように記憶されます。

	X	X
2010	3	1
2011	C	8
2012	2	0
2013	3	E
2014	0	1
2015	0	6
2016	0	0
2017	2	1
2018	0	4
2019	2	0
201A	C	D
201B	B	7
201C	0	2

私たちはこのキットでは256人います。

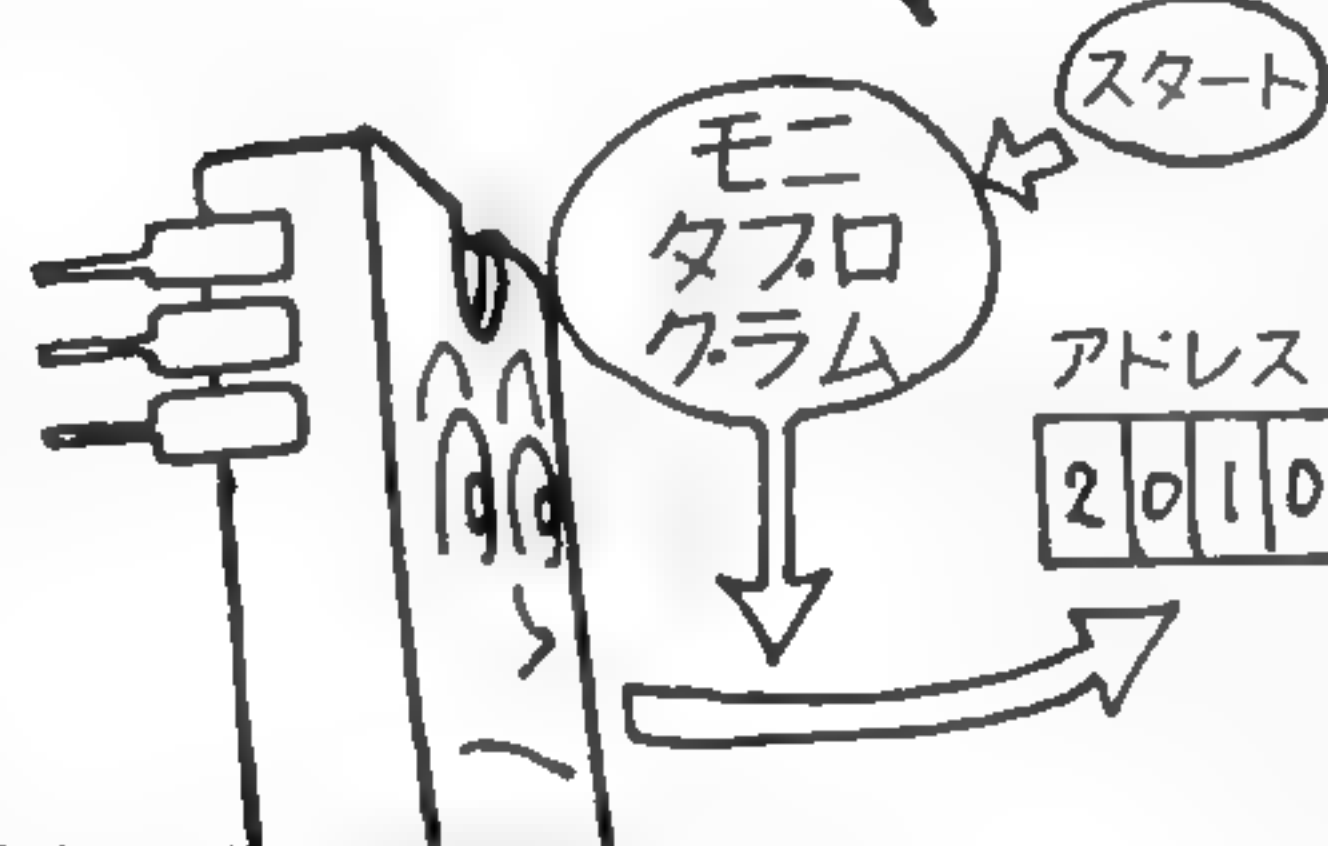
これは、HELPプログラムの最初の部分です。

番地	データ
2010	31C820
2013	3E01
2015	0600
2017	210420
201A	CDB702
...	...

ではこのモニタプログラムを使って私たちはどこまでマシン語を知ることができるのでしょうか。

突然オペレーションコード（略してオペコード、またはOPコード）という言葉が出てきますが、またあとで説明しますので軽く読みとばしてくださいね。

プログラムスタートの時、2010を指定すると、CPUはこの番地のデータをオペレーションコードとして読み込みます。



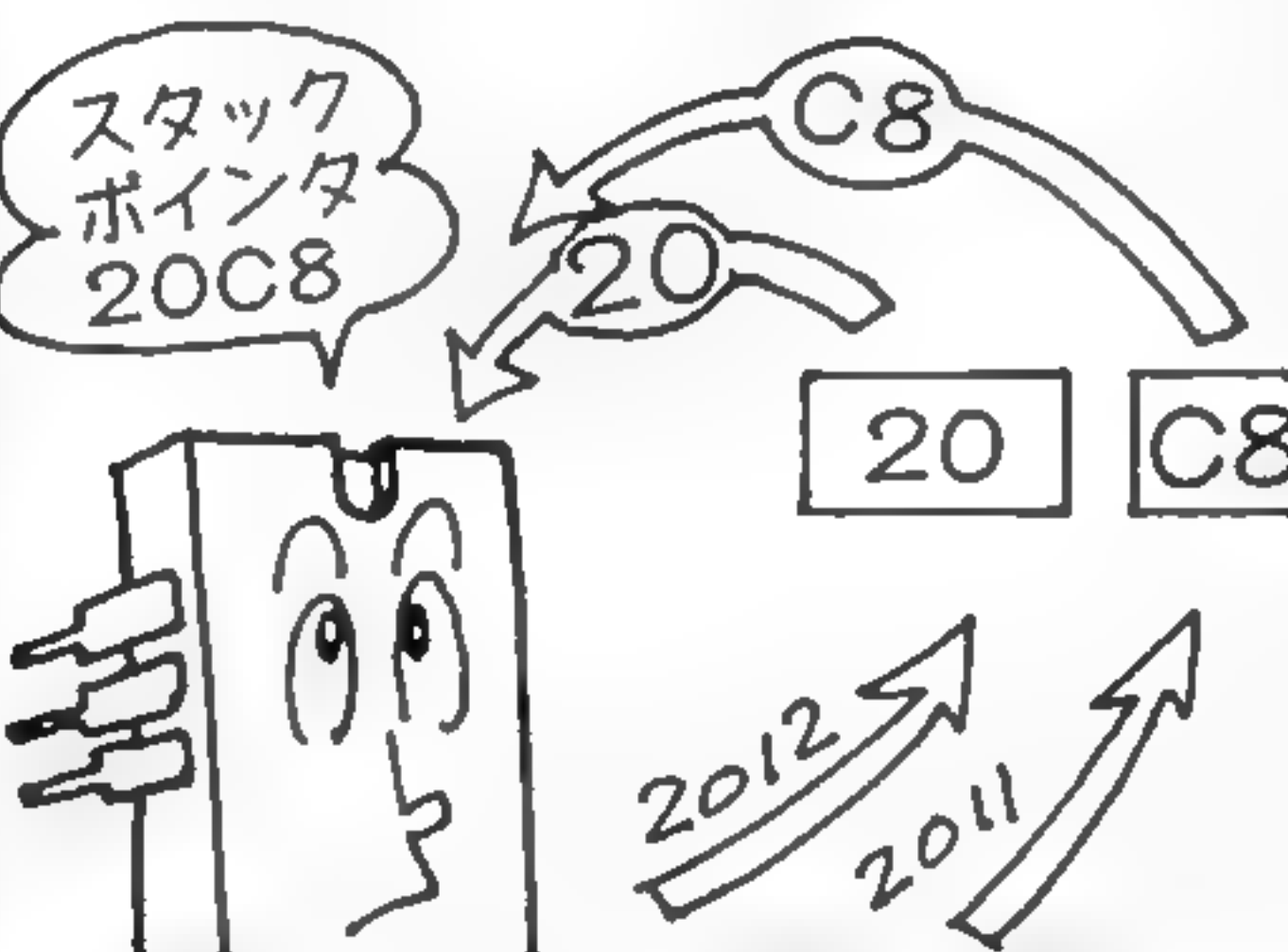
このプログラムでは、2010番地の31というデータがスタートコードになります。



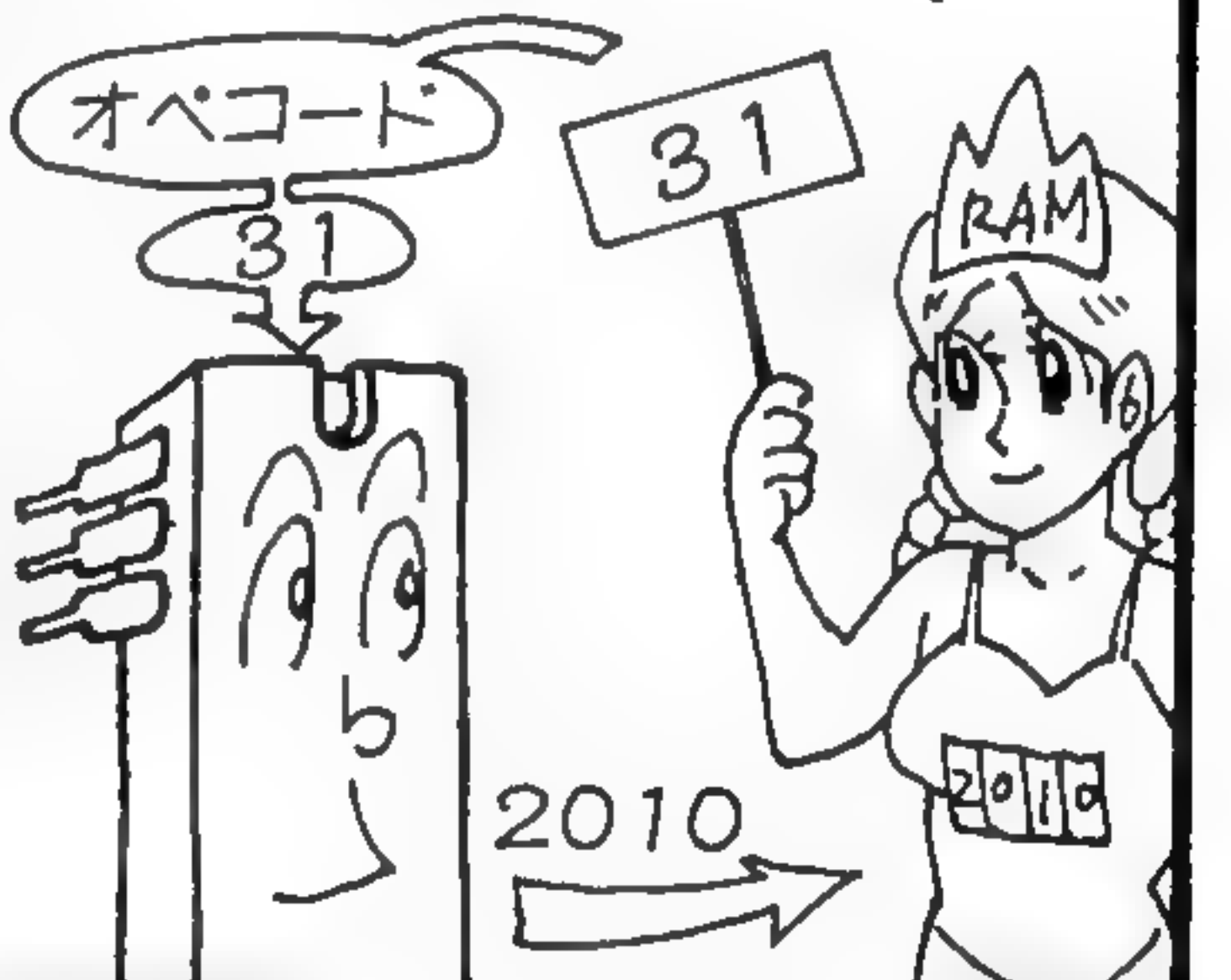
この31というオペレーションコードは、あとに2バイトのデータが必要で、3バイトのセットで使われますので3バイト命令といいます。



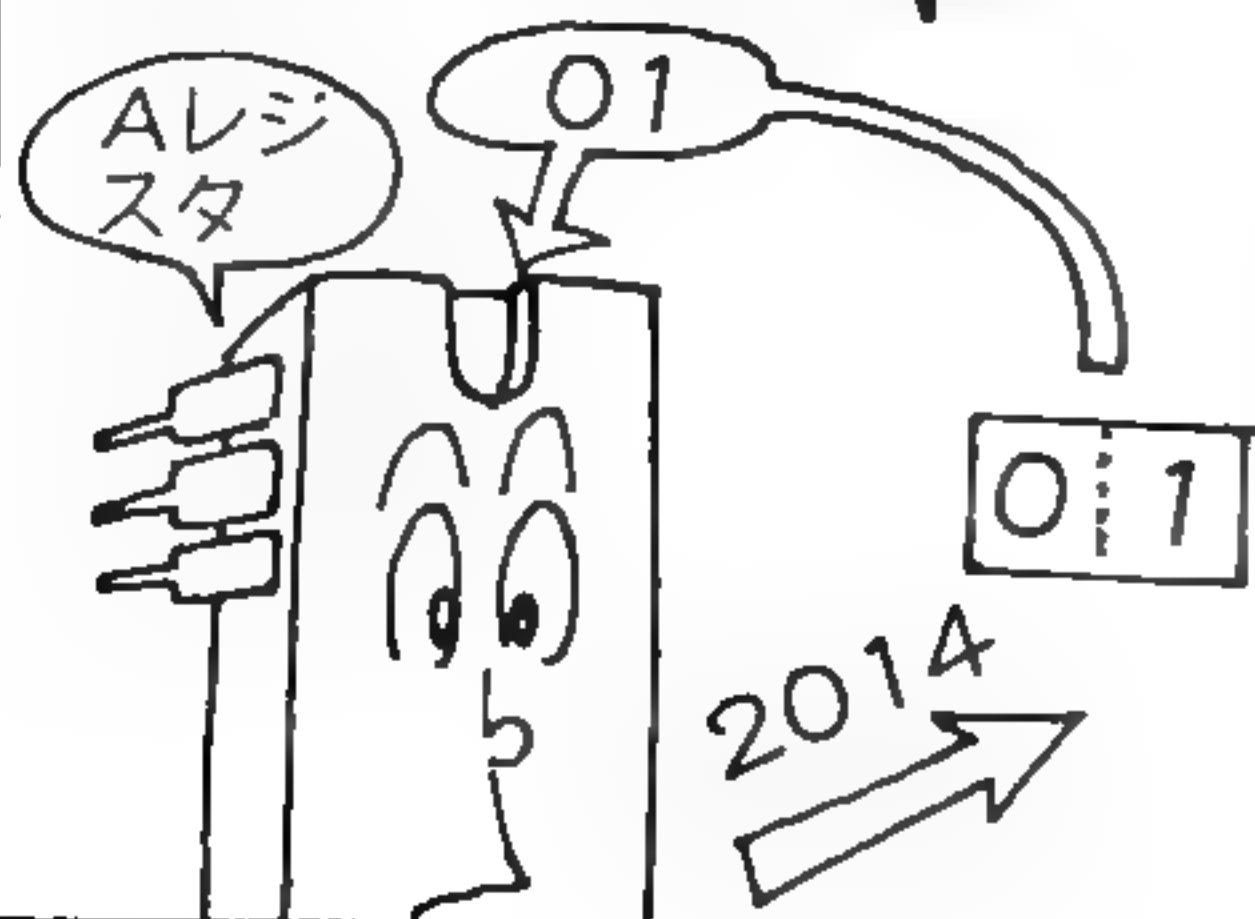
スタックポインタを、あとに続く2バイトで示される番地に設定するという命令です。



2010番地から読み出したデータは31です。このオペレーションコードの意味は、



次に続く1バイトのデータをCPUの中のAレジスタに読み込めという2バイト命令です。あとはこの調子で実行します。



このデータは3Eでオペコードとしての意味は、



このあとオペコードとして読むのは、次のアドレス2013のデータということになります。





モニタプログラムでは、アドレスを1バイト書き込むごとに+1していました。

+1(インクリメント)

でも同じプログラムデータの中で全部が全部CPUへの命令でないことはなんとなくおわかりでしょう。

番地	データ
2010	31 C8 20
2013	3E 01
2015	06 00
2017	21 04 20

CPUへの命令コード      そうでないコード

今の説明ではスタックポイントなどの言葉がでてきてチンプンカンプンだったかもしれませんが。

そのため、プログラムデータは、これを打ち込む人間の方で十分に注意する必要があります。

本当のオペコードかただのコードかの区別はできません。

OPコード   OPコード   OPコード

31 08 20 3E 01

2010

CPUは、最初のOPコード解読から次々に連続してデータを読み込んでいます。

31は3バイト命令

CPU

メモリ

OPコード一覧表

このことはROM型パソコンのMONモードでも同じことが言えます。暴走すると、プログラムが崩れてしまう恐れがあるため事前にLコマンドを使ってテープにしまっておく方が無難ですよ。

HELPプログラムは例題としてメーカーの人が作っておいてくれたもので、正しくインプットすればすぐ実行できます。

もし途中、OPコードを間違えて入れたり、データが不連続だったりすると、実行時正しく働きません。

.....

.....

暴走中

自分でこのマシン語を使ってプログラムしてみたい時はいったいどうしたらいいのでしょうか？それにはCPUのしくみを知らなくてはなりません。

ところで31C8203E...とならんでいる数字がマシン語らしいということは、よくわかったのですが、

マイコン雑誌などに載っているマシン語リストを見ながらキーインする時は数字ひとつも間違えないように注意深く行なうことです。

FM-7'8用   バイナリー   インベータ

4000   10   CE   6F   FF   86

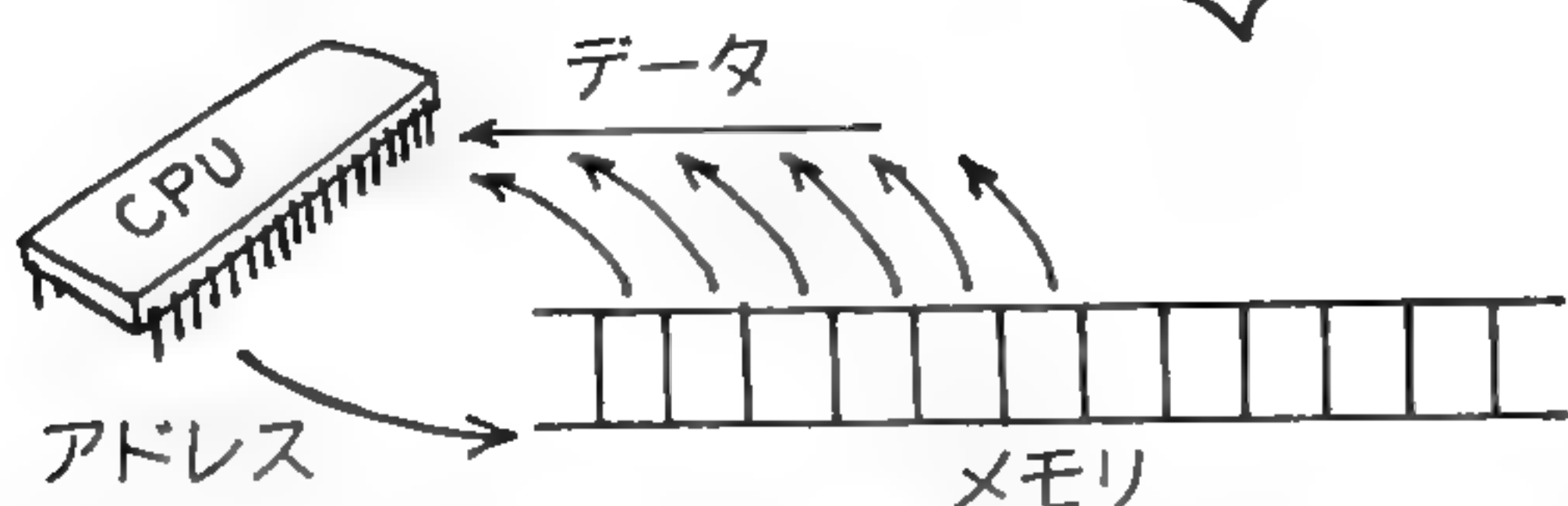
4010   4B   20   17   00   D

96   3D   2A   1E

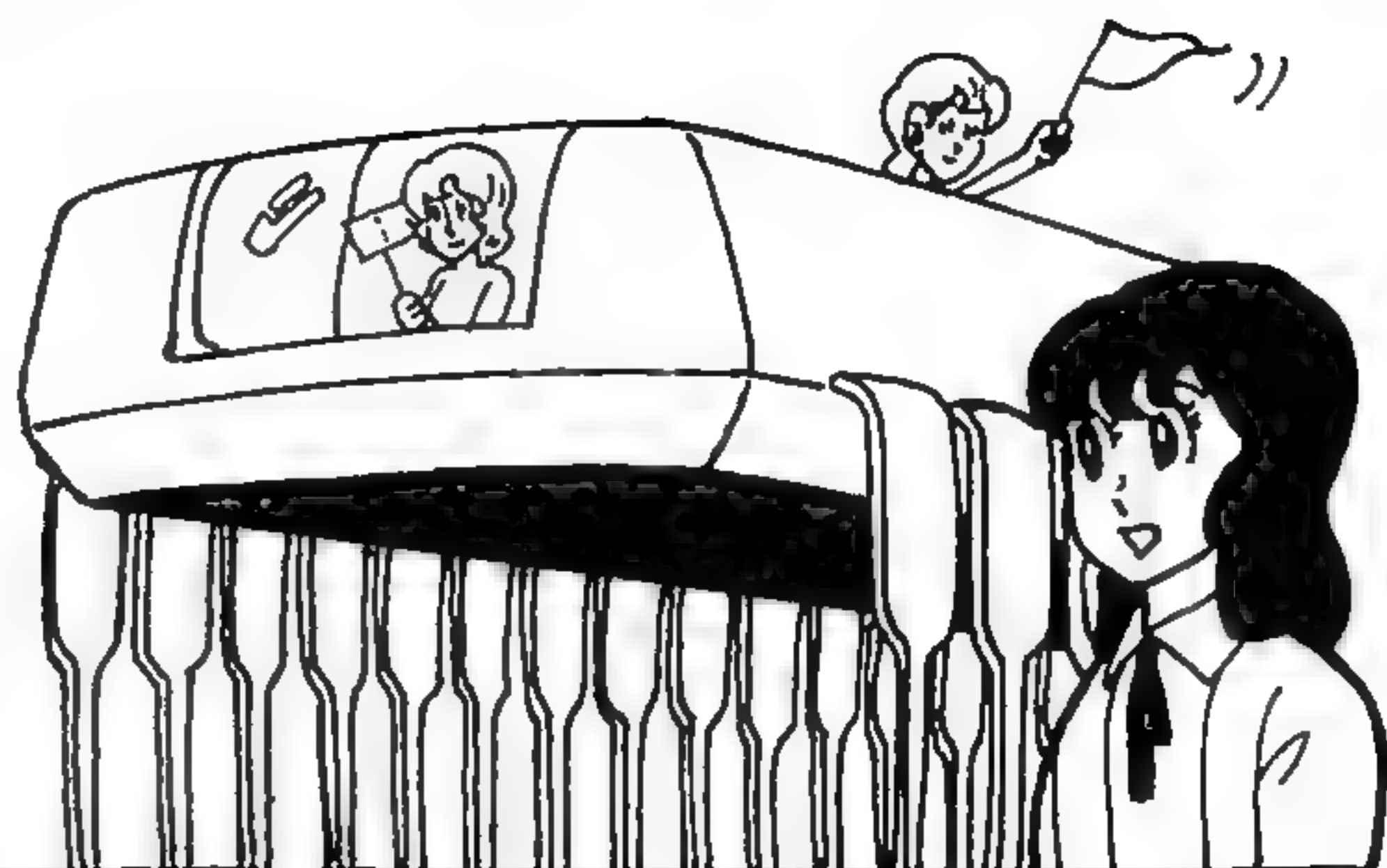
DB   3C   D7



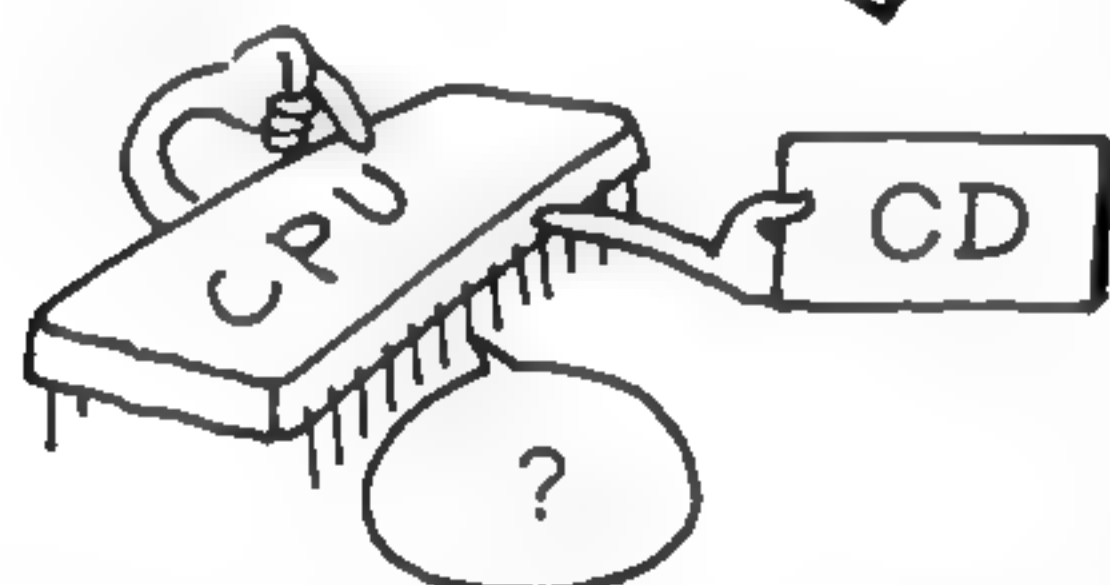
CPU というのはメモリに入っているデータを読み込んでその内容を解読して所定の処理を行ない、また次のメモリに入っているデータを読む……といったことをくり返すようにできています。



## ④CPUのしくみ (8080の場合)



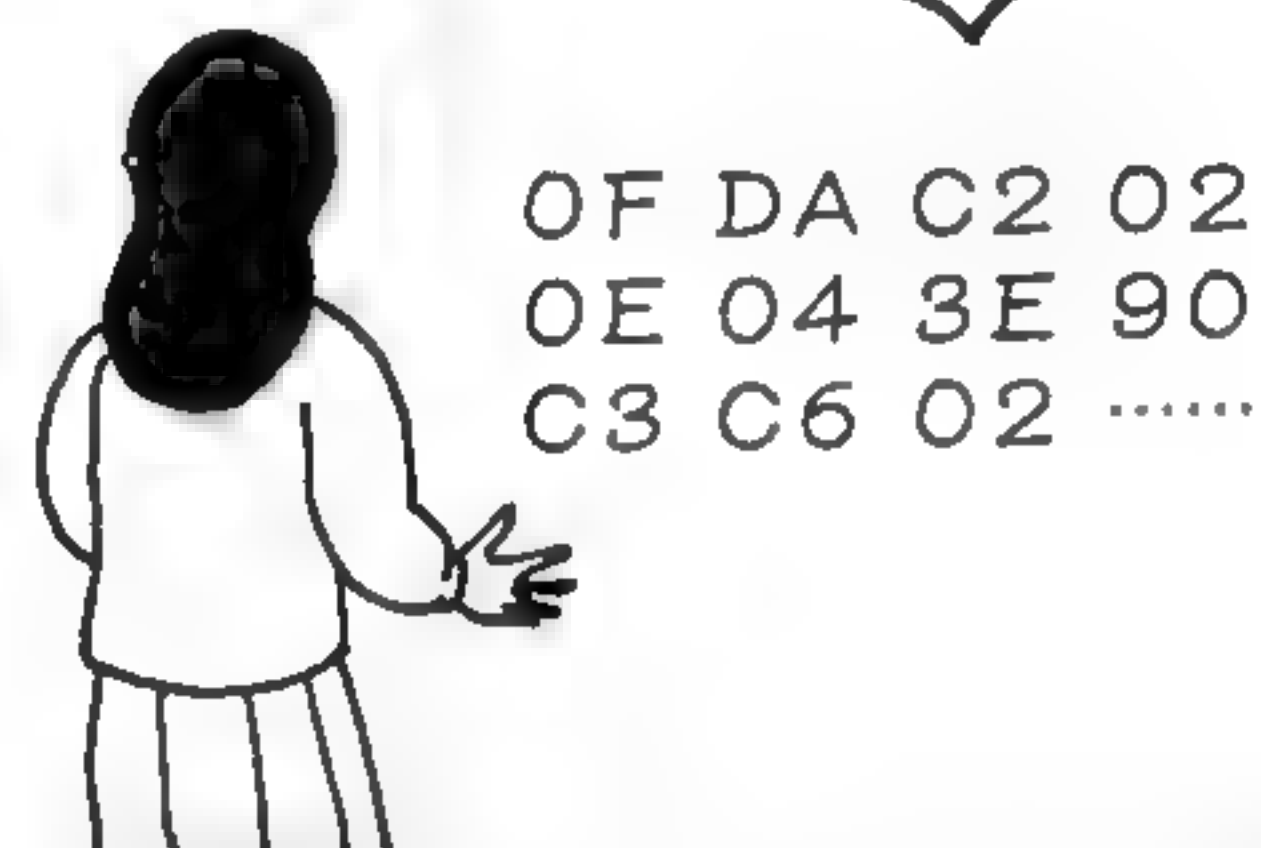
そのためには00、01、02……FD、FE、FFといったコードがCPUにとってどんな意味を持つものなのかを知らなくてはなりません。



このように、うまくデータをメモリの中にならべておくことを、フロッグミングと言っているわけです。



そこでメモリの中に入れるデータを工夫すれば、CPUがあたかも人工頭脳のような働きをするのです。



0F DA C2 02  
0E 04 3E 90  
C3 C6 02 .....

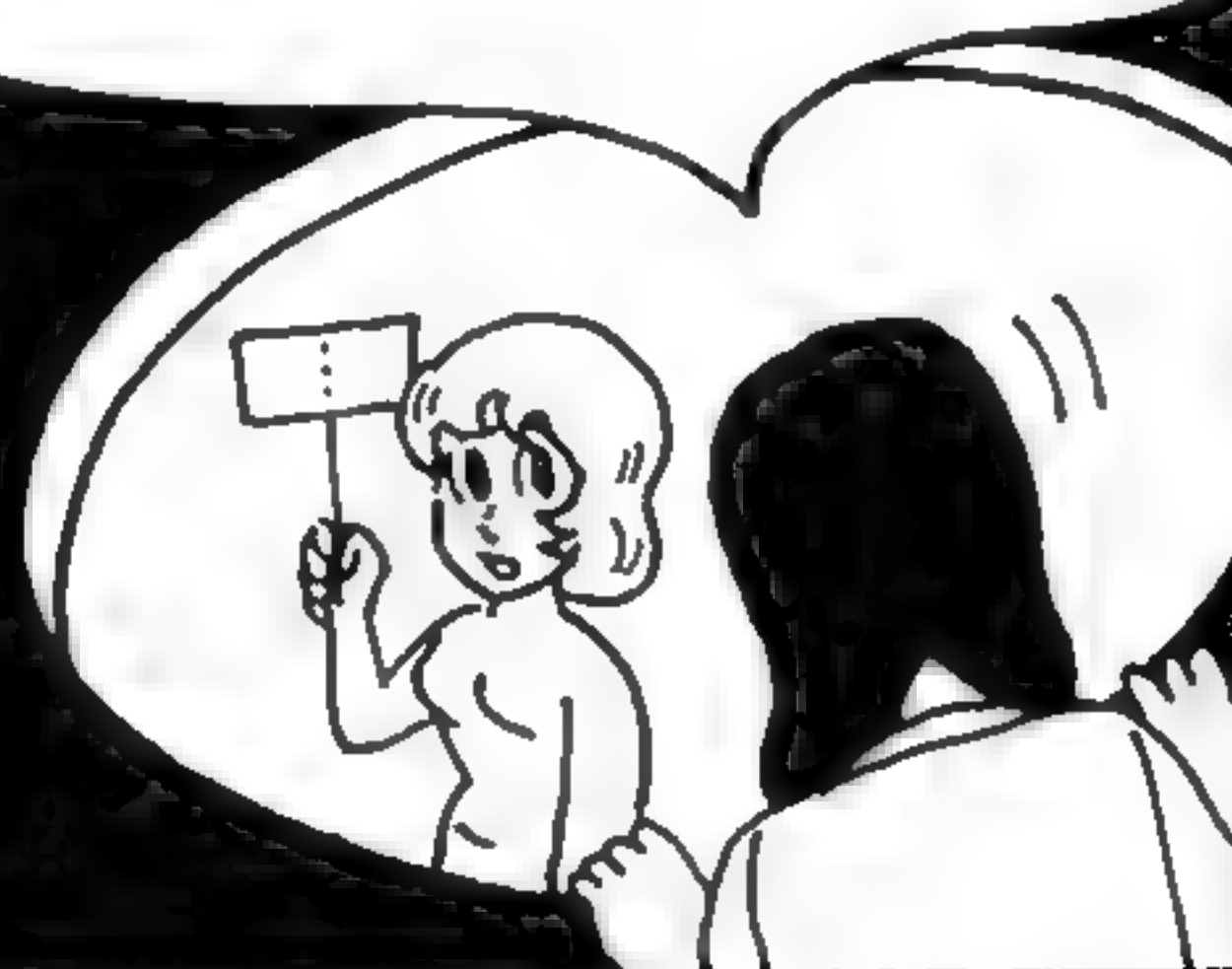
これらの働きや役目がチンファンクションでは、とうていマシン語のプログラムは組めません。

MVI C, 4

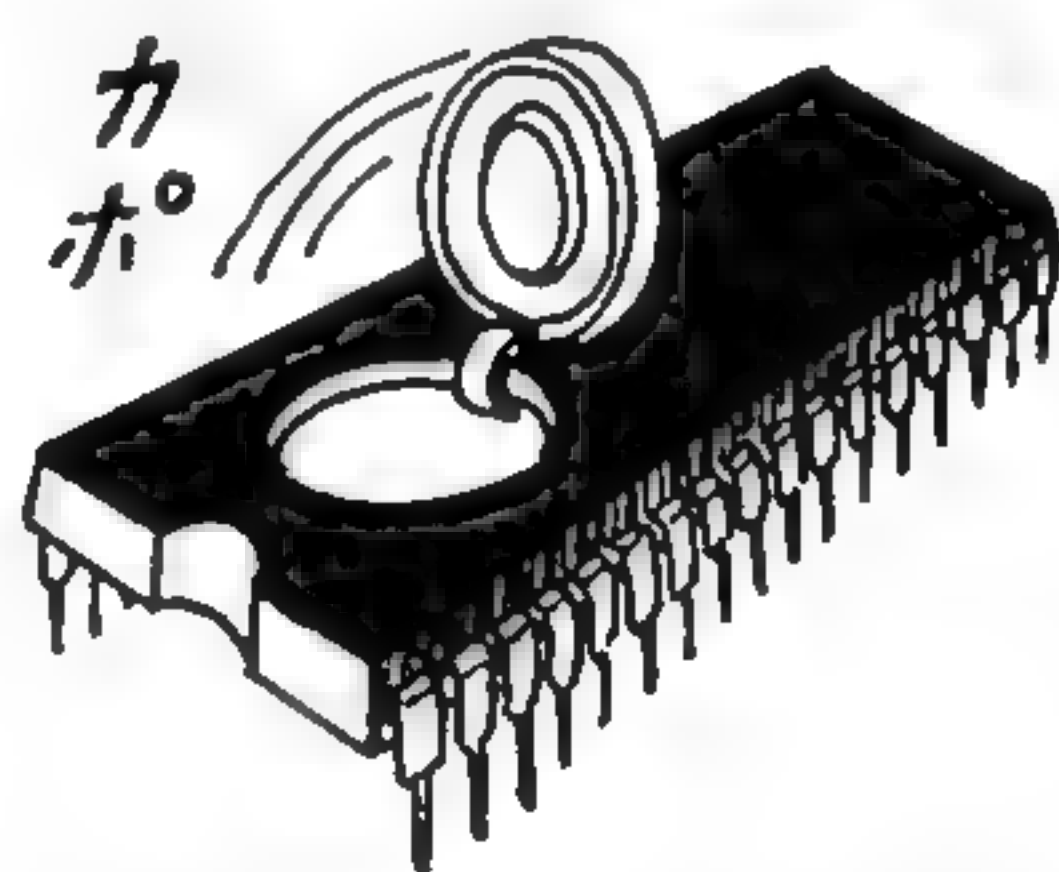


Cレジスタに4を入れる

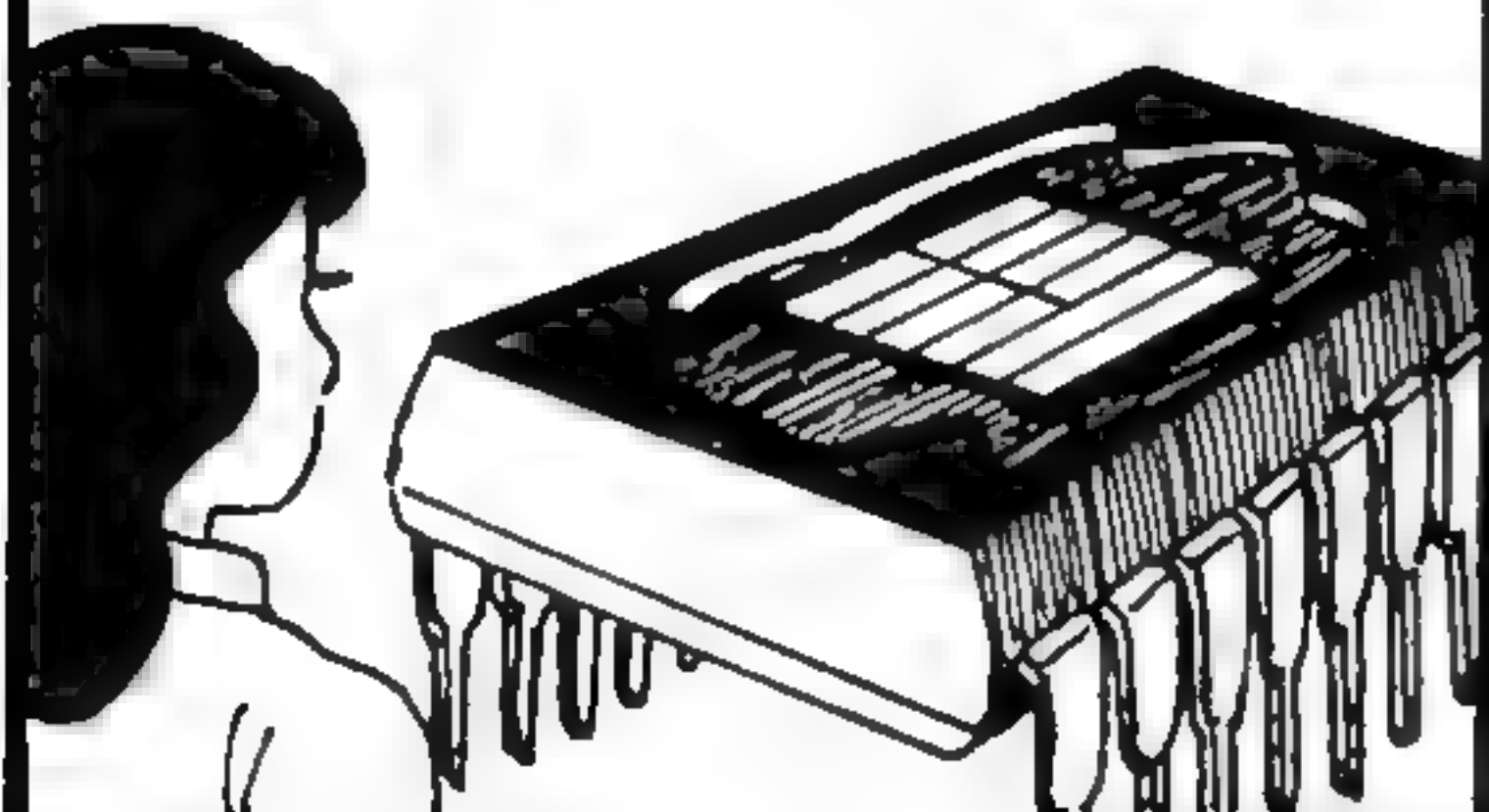
たとえばレジスタ、プログラムカウンタ、スタックポインタ、フラグなどはCPUの中にあるものです。



また、CPUの中であって、プログラムする上でどうしても知っておかなくてはならないこともあります。

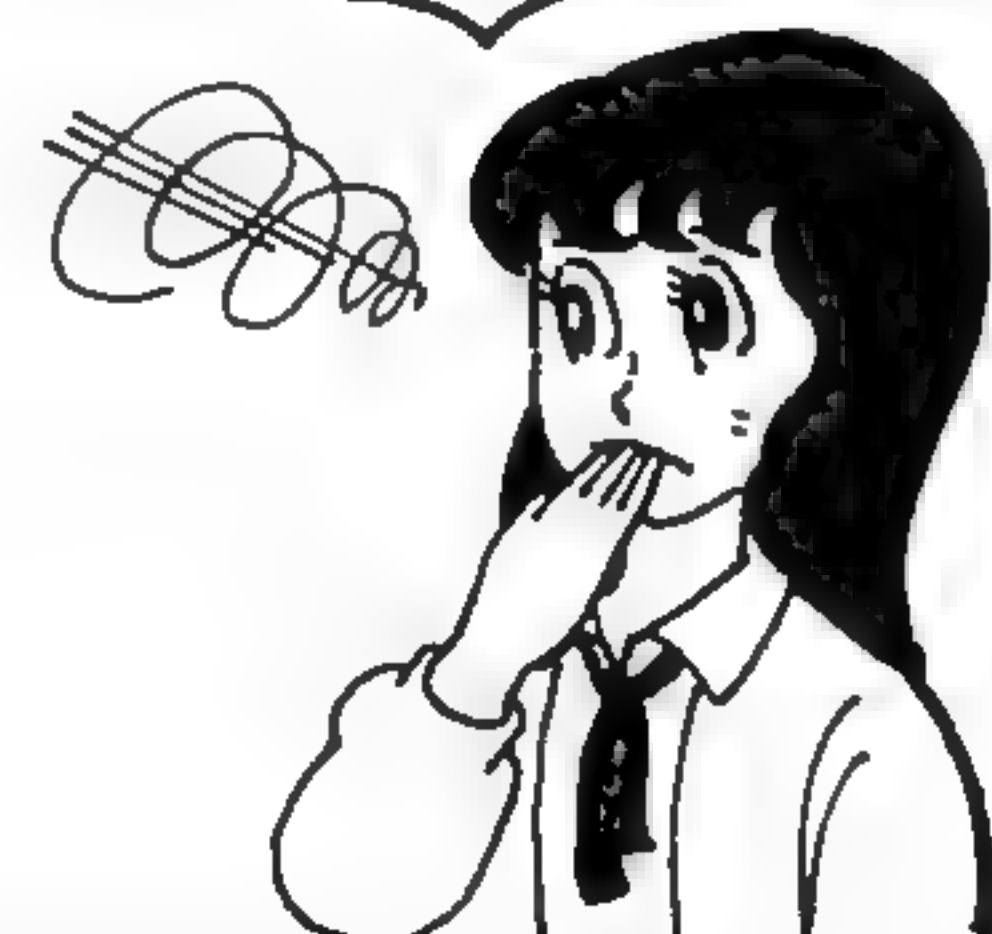


そこでもう少しマンガチックに表現してみました。CPUの中であってプログラムに関係があるものです。



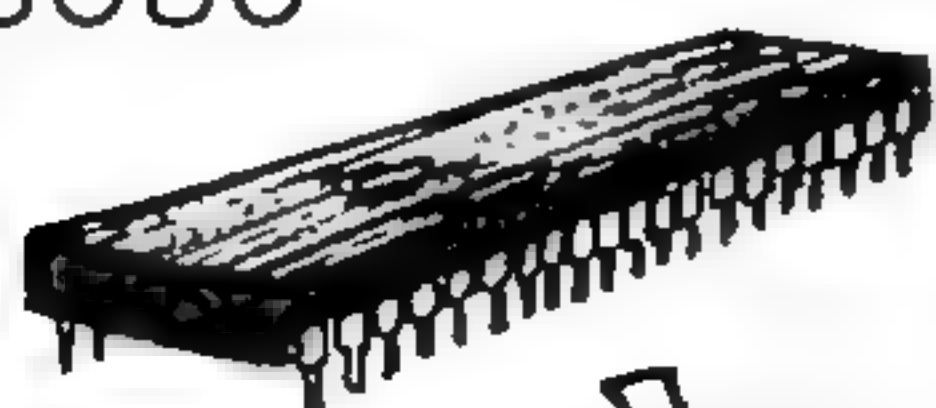
8ビット		8ビット	
A		F	F
B		C	
D		E	
H		L	
PC			
SP			
16ビット			

マニュアルなどにはこのように書いてありますがチンファンクションもいいところ……。





8bitマイクロコンピュータ  
8080



アキュムレータ  
(レジスタ)

CPUの中を  
のぞいてみ  
ると…。

レジスタたちが、CPUの中のどこにいるのかわかっておくことはソフト的な機能を知るうえでとても参考になります。

あ A ちゃん

ち  
ち

15

二

せ

757"  
7リッ7° 70~7°

01... 00  
02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600 601 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627 628 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646 647 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664 665 666 667 668 669 670 671 672 673 674 675 676 677 678 679 680 681 682 683 684 685 686 687 688 689 690 691 692 693 694 695 696 697 698 699 700 701 702 703 704 705 706 707 708 709 710 711 712 713 714 715 716 717 718 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733 734 735 736 737 738 739 740 741 742 743 744 745 746 747 748 749 750 751 752 753 754 755 756 757 758 759 760 761 762 763 764 765 766 767 768 769 770 771 772 773 774 775 776 777 778 779 780 781 782 783 784 785 786 787 788 789 790 791 792 793 794 795 796 797 798 799 800 801 802 803 804 805 806 807 808 809 810 811 812 813 814 815 816 817 818 819 820 821 822 823 824 825 826 827 828 829 830 831 832 833 834 835 836 837 838 839 840 841 842 843 844 845 846 847 848 849 850 851 852 853 854 855 856 857 858 859 860 861 862 863 864 865 866 867 868 869 870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887 888 889 890 891 892 893 894 895 896 897 898 899 900 901 902 903 904 905 906 907 908 909 910 911 912 913 914 915 916 917 918 919 920 921 922 923 924 925 926 927 928 929 930 931 932 933 934 935 936 937 938 939 940 941 942 943 944 945 946 947 948 949 950 951 952 953 954 955 956 957 958 959 960 961 962 963 964 965 966 967 968 969 970 971 972 973 974 975 976 977 978 979 980 981 982 983 984 985 986 987 988 989 990 991 992 993 994 995 996 997 998 999 1000 1001 1002 1003 1004 1005 1006 1007 1008 1009 1010 1011 1012 1013 1014 1015 1016 1017 1018 1019 1020 1021 1022 1023 1024 1025 1026 1027 1028 1029 1030 1031 1032 1033 1034 1035 1036 1037 10

8ヒット内部データ・バス

私たちはアツ  
アツカッフル。  
くつついたり、  
はなれたり……

汎用  
レジスタ

データは  
保護が仕  
事です。

私は、フロクラムの進行係です。

プログラ  
カウンタ

スタック // ポインタ

↓

PC

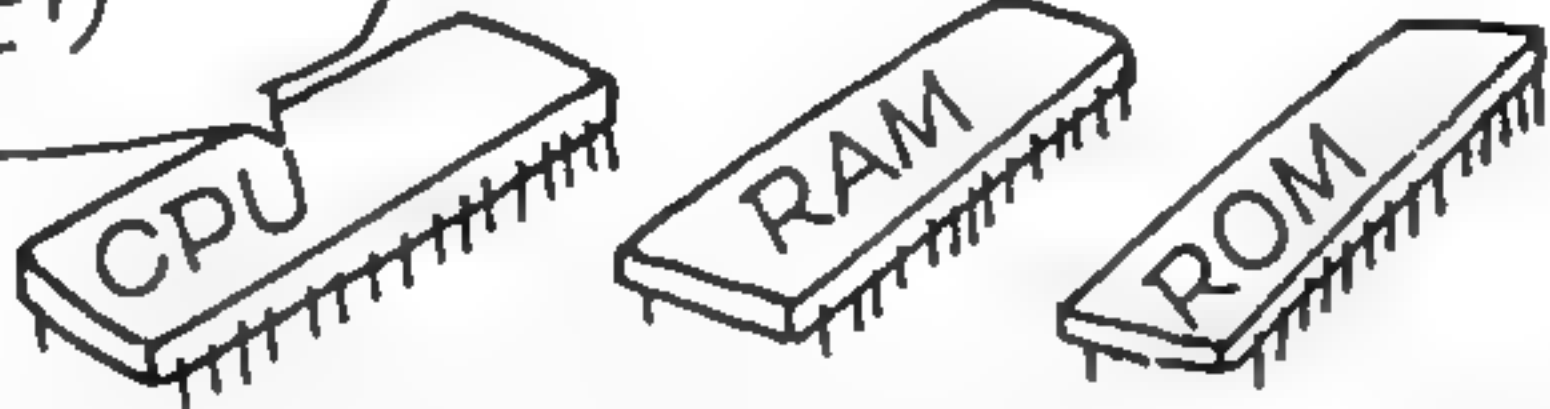


# ⑤マシン語のしくみ

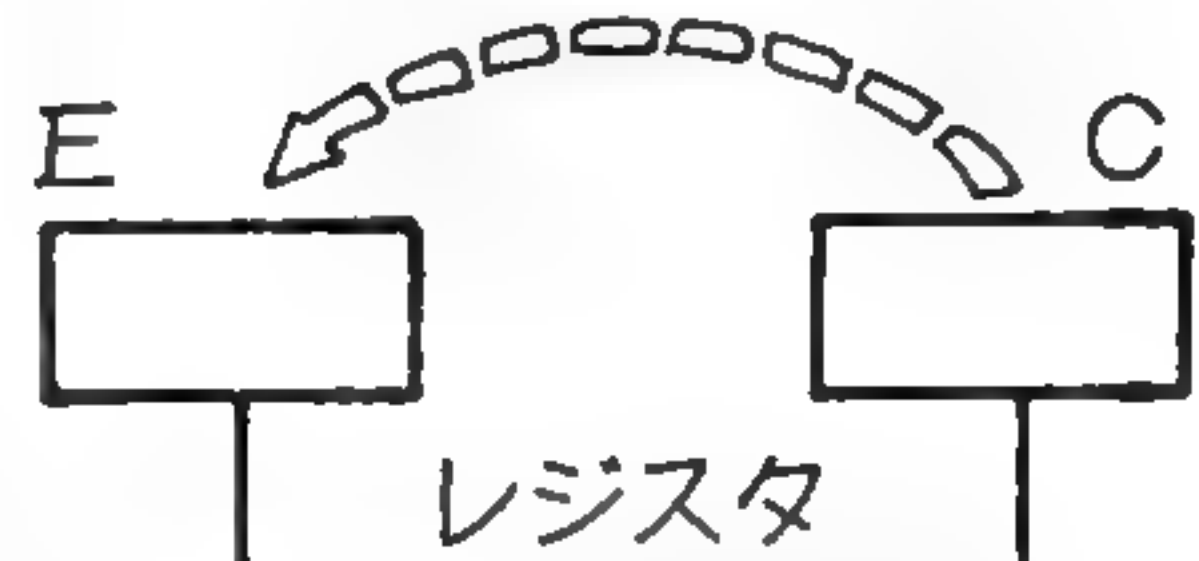
レジスタというのはCPUの中で一時的にデータを記憶する場所のことです。データとして8ビット、アドレスとして16ビット記憶できるように工夫されています。

レジスタはCPUの内部メモリ

メモリ

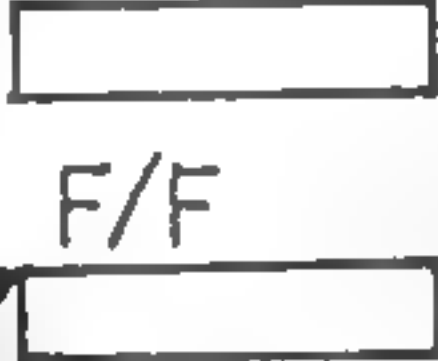


MOV E, C

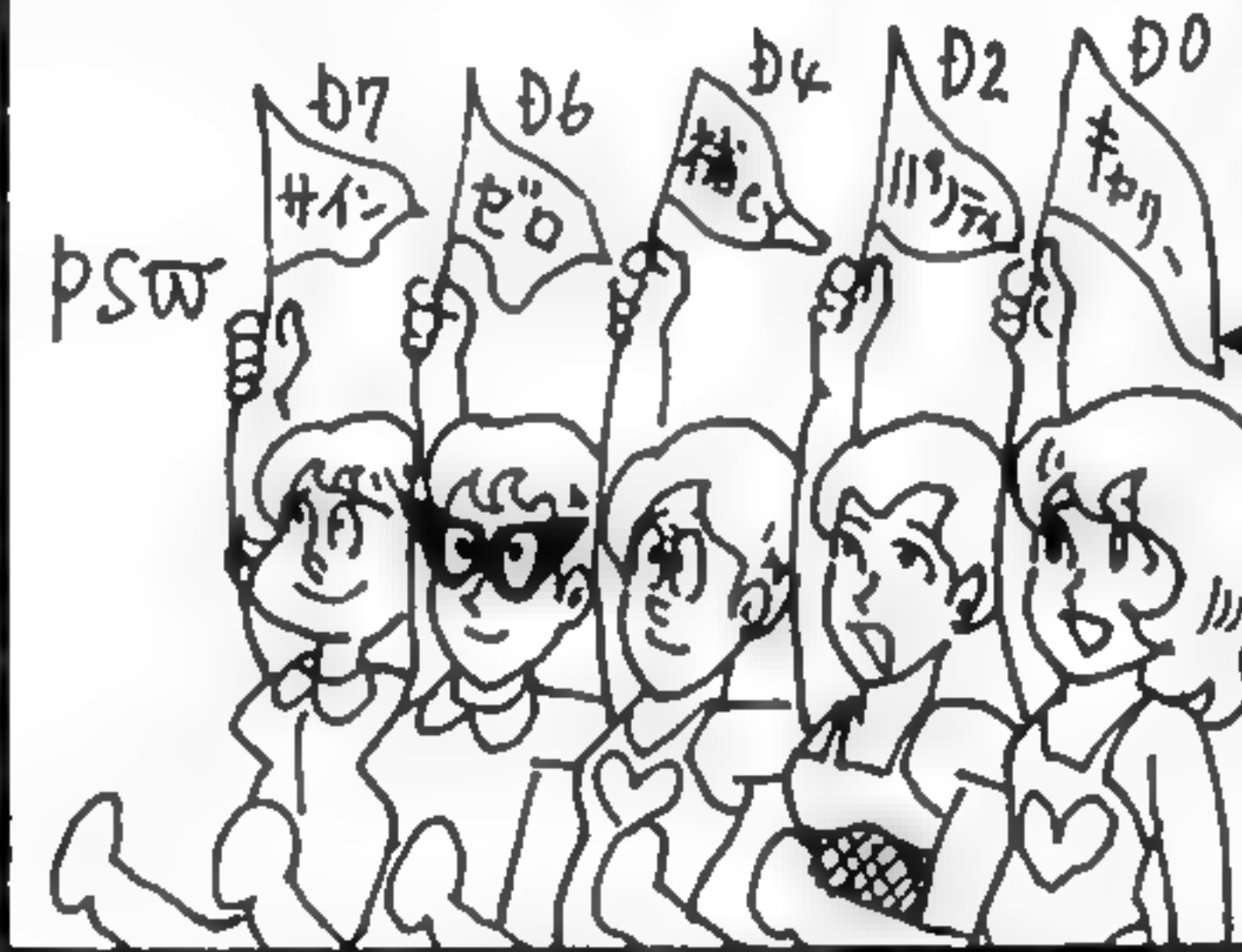


このアキュムレータとフラグの関係は、とてもコンピュータ的です。その話はまたあとでね。

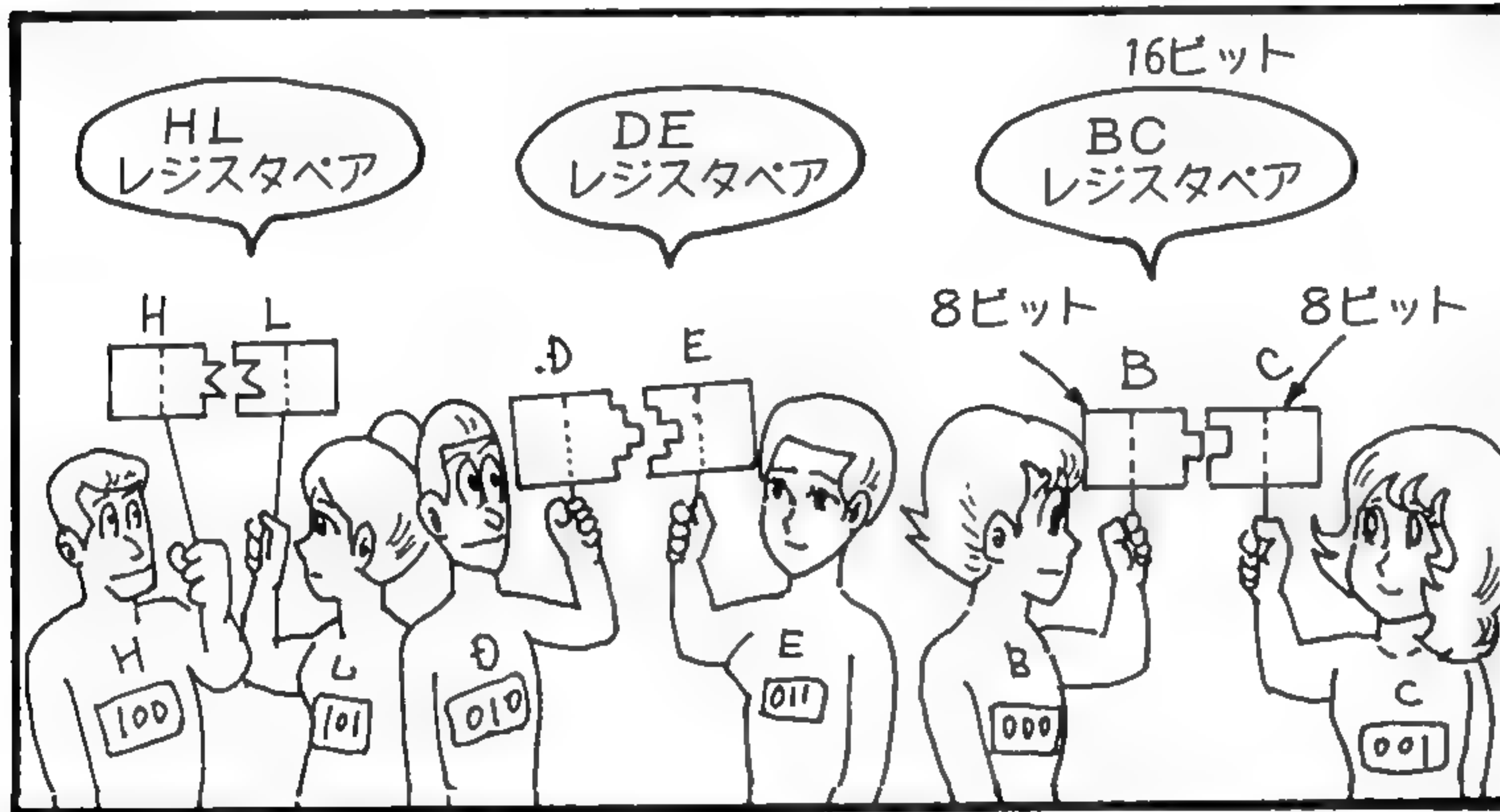
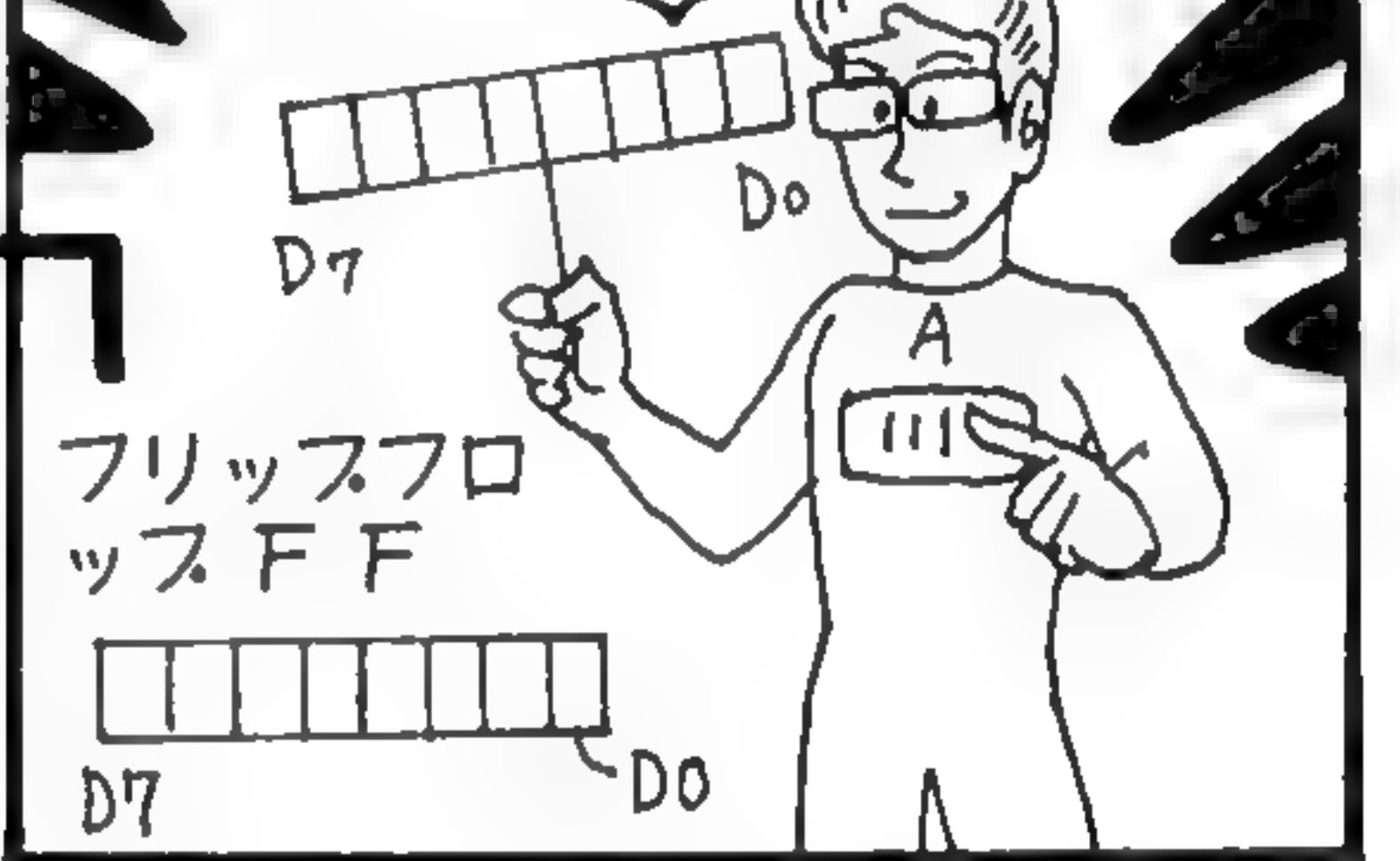
AまたはAcc



私たちフリップフロップ(F/F)は、フラグビットの集まりです。1ビットごとに意味があり、プログラムで使います。



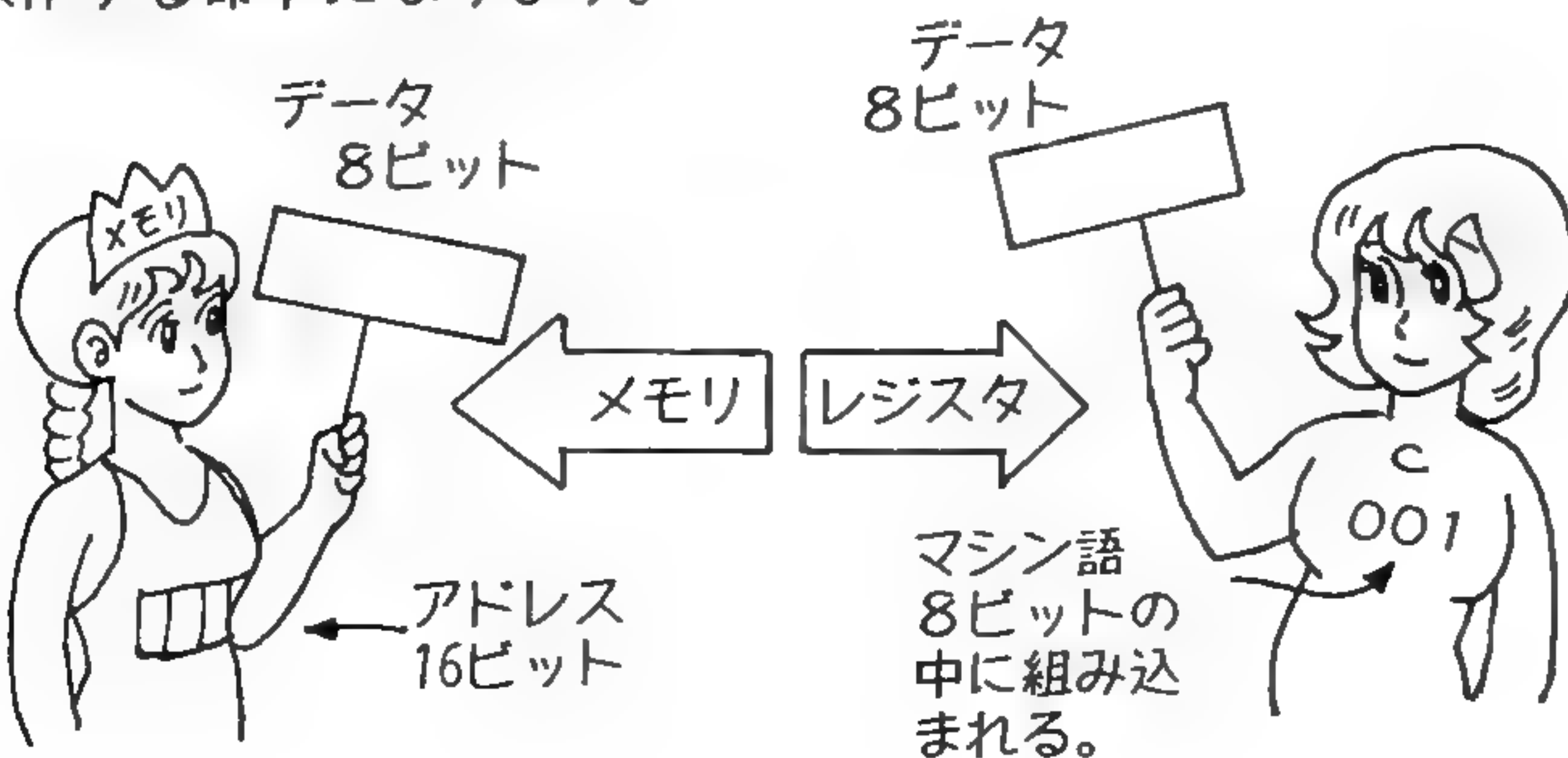
僕はアキュレータ。8ビットのレジスタとしても使われますのでAレジスタとも呼ばれます。メモリのアドレスに相当するコードは3ビット111です。



B、C、D、E、H、Lの6つのレジスタはそれぞれペアを組むことができます。



レジスタにはROMやRAMのようにアドレスがありません。しかし、それぞれ3ビットのコードを持っていてこれがマシン語の中に組み込まれ、そのレジスタを操作する命令になります。





たとえば、レジスタ間のデータをやりとりする（転送という）命令は下のような形式で行なうように決められています。

この01がレジスタ間データ転送を意味する。

01 DDD SSS

データを受け取る側のレジスタコードを入れる。

8ビットをこのように意味づける。

データを送る側のレジスタのコードを入れる。

8080の場合、マシン語のコードは00～FF(256通りある)で表現されますが、適当に意味をつけているわけではありません。



- 00……何もせず次に進む
- 01……Bレジスタに2バイト入れる
- 02……(B)番地のデータをAレジスタに入れる
- 03……(BC)をインクリメント

もちろん01001111なんていちいち書いていられないので、4ビットずつ2つに区切って4F（ヨンエフ）と表現します。

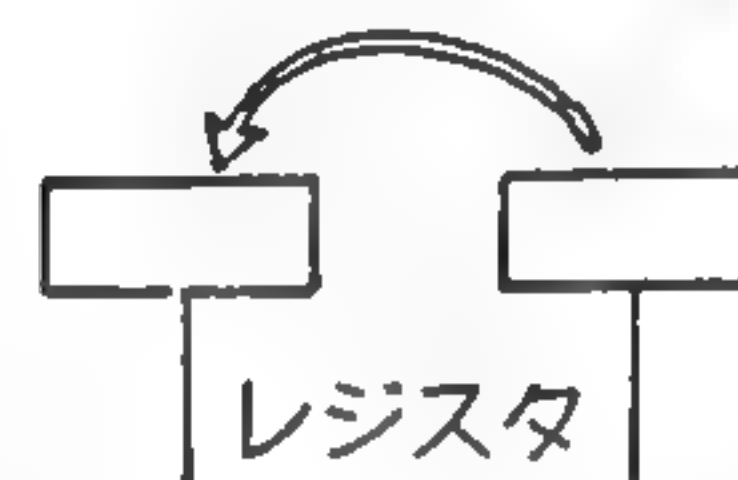
4 F  
01001111

たとえばAレジスタの内容をCレジスタに送るマシン語はAのコード111とCのコード001を入れて、このようになります。

01 001 111  
Cレジスタ Aレジスタ

だから、この01がレジスタ間データ転送であることを覚えているだけで、マシン語を作れるのです。

01……



今Aレジスタに47というデータが入っていたとして4Fというマシン語の命令が実行されると、Cレジスタに47が入ります。私たちはこの動きを4Fというコードから読みとることができません。



マニュアルやマイコン雑誌でみるマシン語はすべてこのように1バイト（8ビット）のデータを2つに区切って16進数で表現したものです。



8080では、この場合、MOVがニーモニックになります。

スペース コンマ  
MOV C, A=4F  
↑ ↑  
受ける側のレジスタ名 送る側のレジスタ名

そこでこのマシン語の意味を表現できるように考えたのがニーモニックコードなのです。

英数字表現

16進表現

ニーモニックと言う



つまりマシン語をつくることはできてもあとでそのコードを見て動作内容を思い出せるかどうかは問題です。

?

4F

えーと、えーと。忘却とは…。

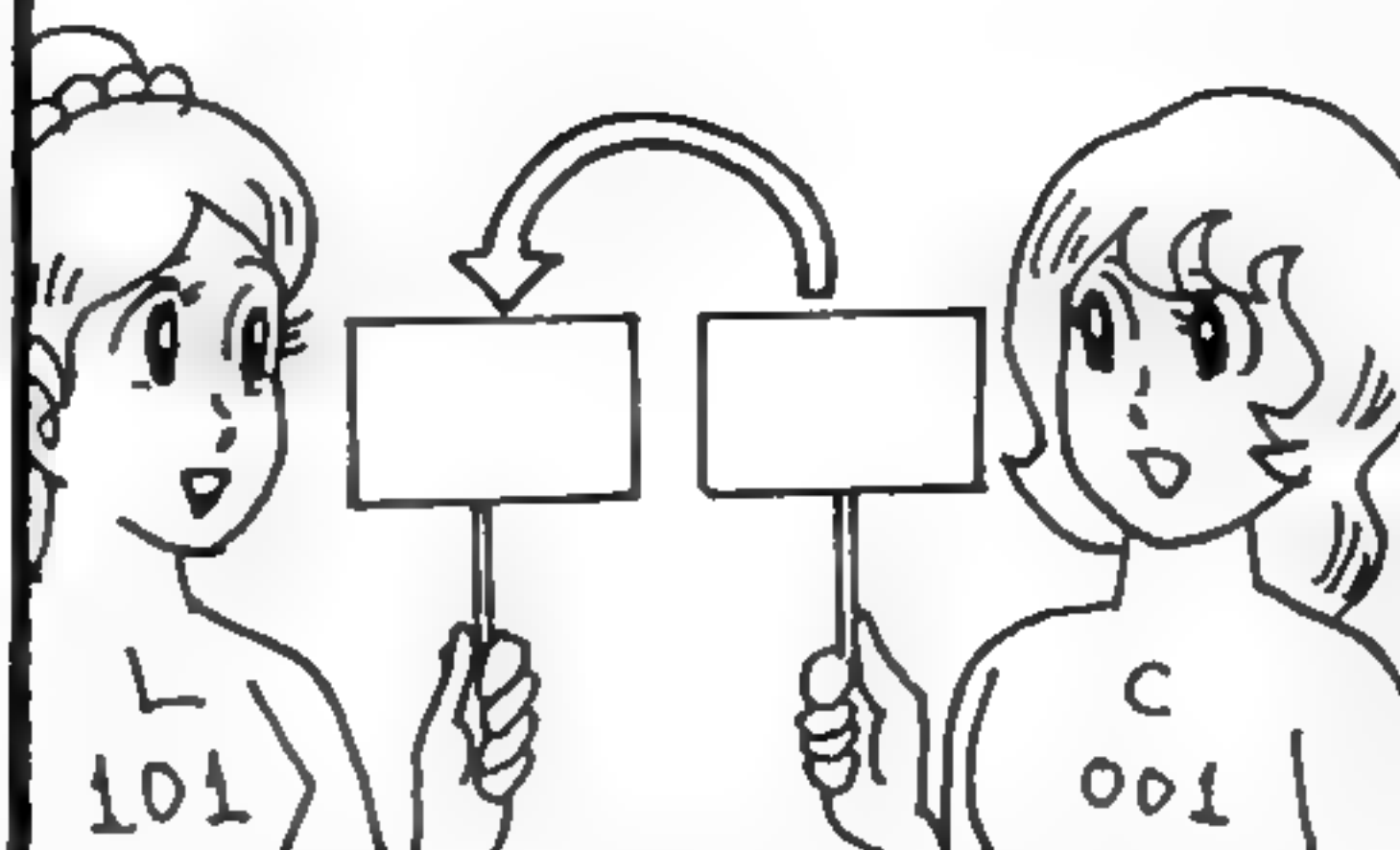




マシン語はLレジスタのコードが101ですから、さっきと同様に作れます。69 (ロクキューです)。

6 9  
01 101 001  
L C

動作はCレジスタの内容をLレジスタに転送することです。

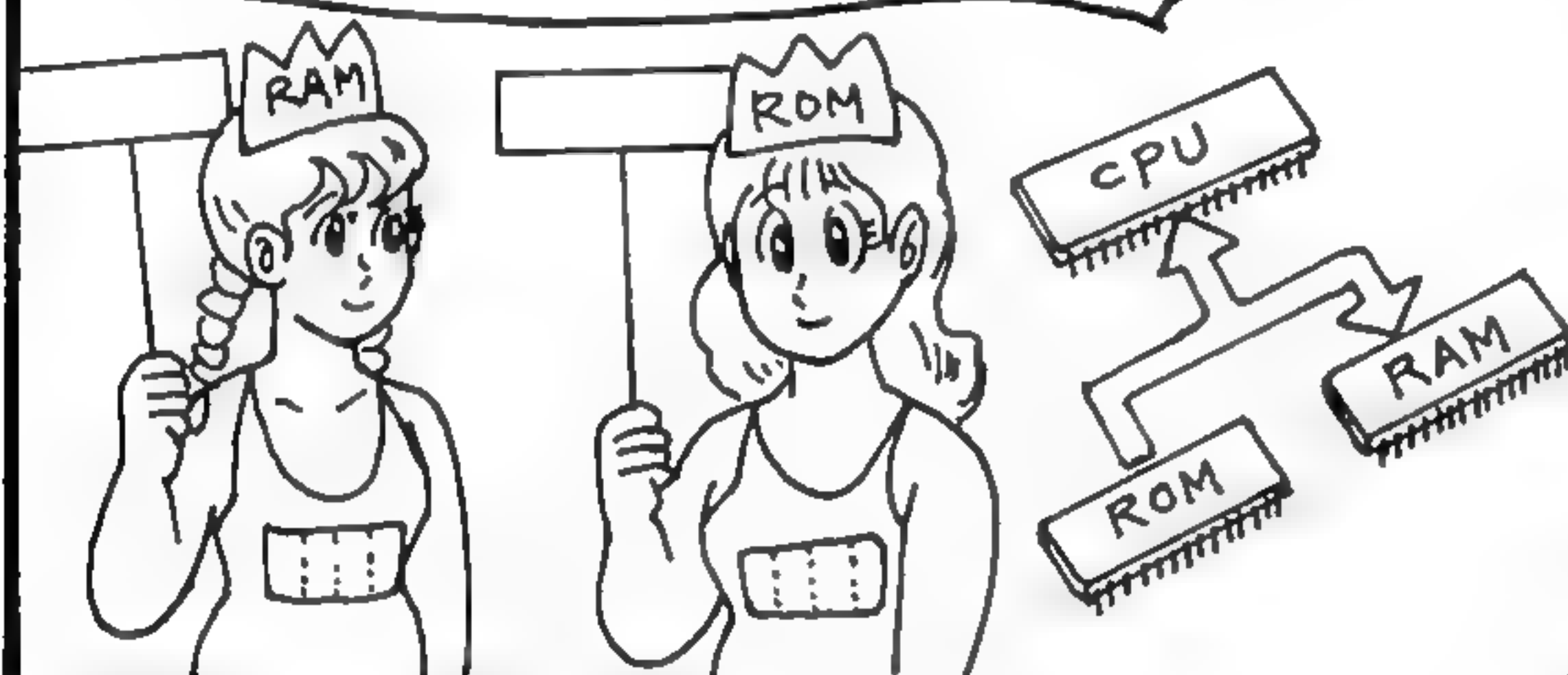


では、MOV L, Cという命令の動作とマシン語はどうか見当がつきますか？

MOV L, C



次はCPUの外のメモリとレジスタ間のデータ転送を考えてみましょう。ハード的にはROMちゃんとRAMちゃんの2種類ありましたね。



レジスタ間のデータ転送命令はこの調子でどんどん作れます。

MOV B, B  
MOV C, B  
MOV D, B  
...  
MOV B, C  
MOV C, C  
...

( )内はレジスタコード

A(111)	—
B(000)	C(001)
D(010)	E(011)
H(100)	L(101)

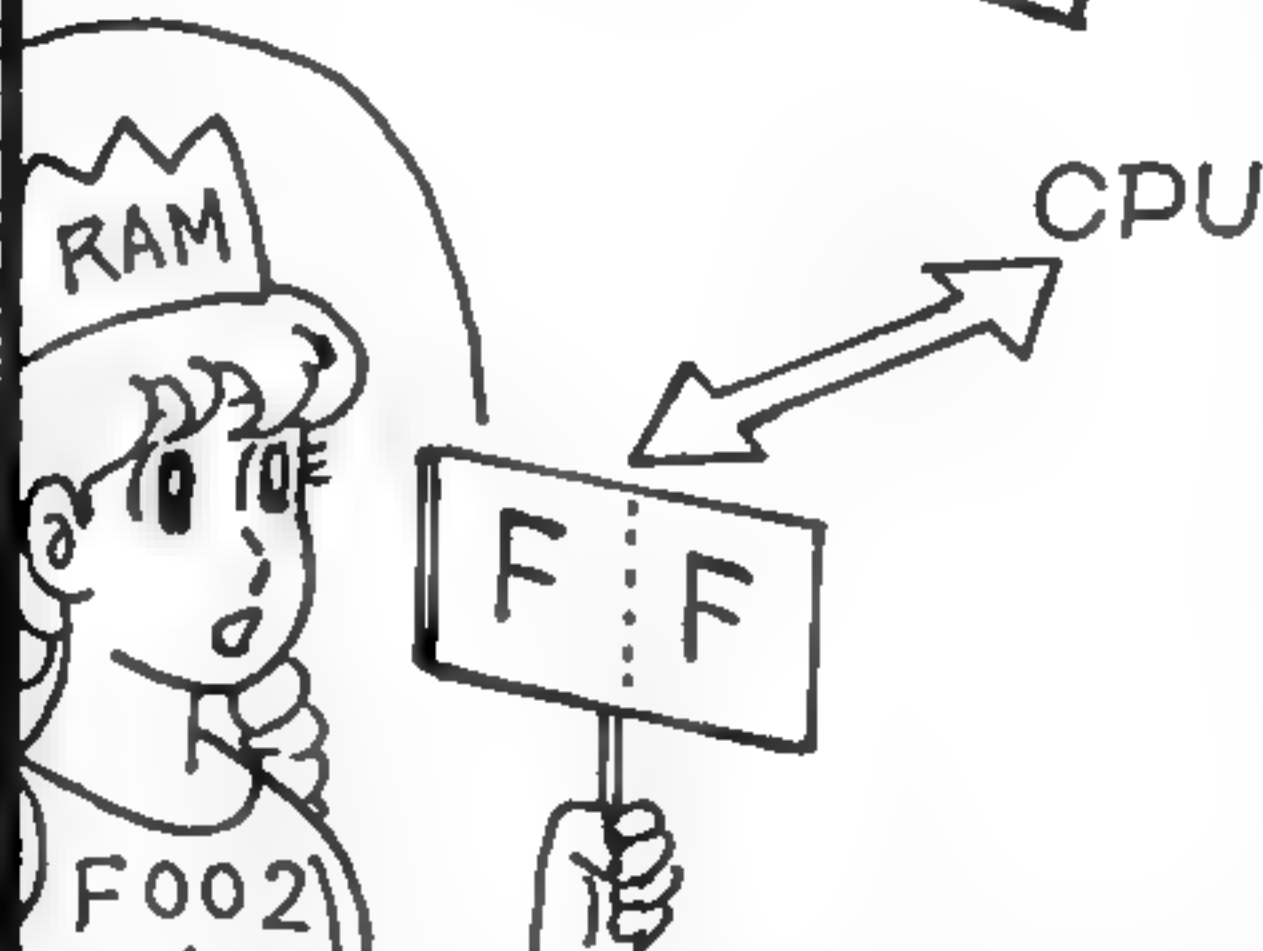
M(110)

どちらもアドレスがあらかじめハード設計の時に決められています。

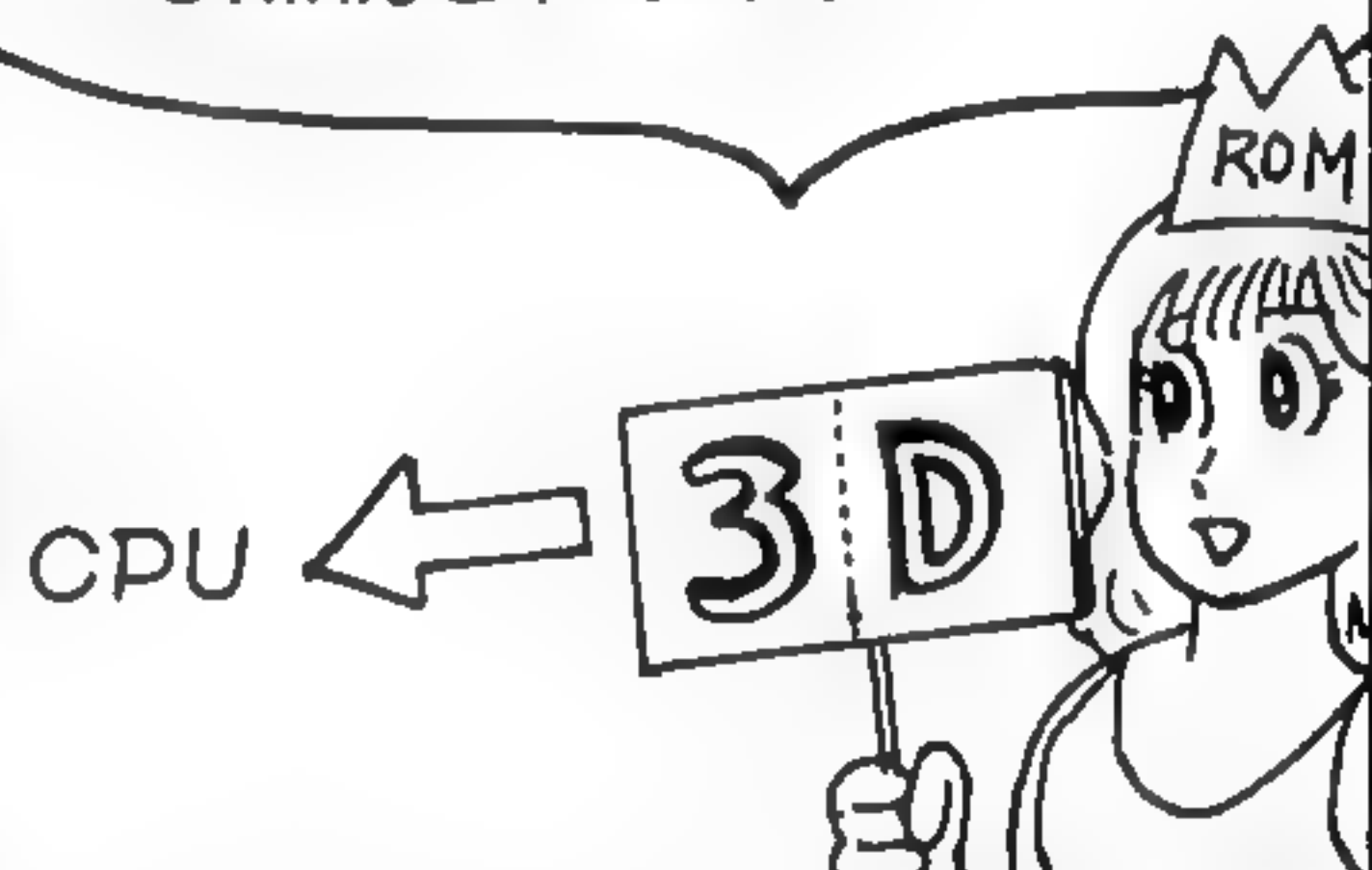
中身

F000	2C
F001	3D
F002	FF
F003	AB
F004	53
F005	4C
F006	B5

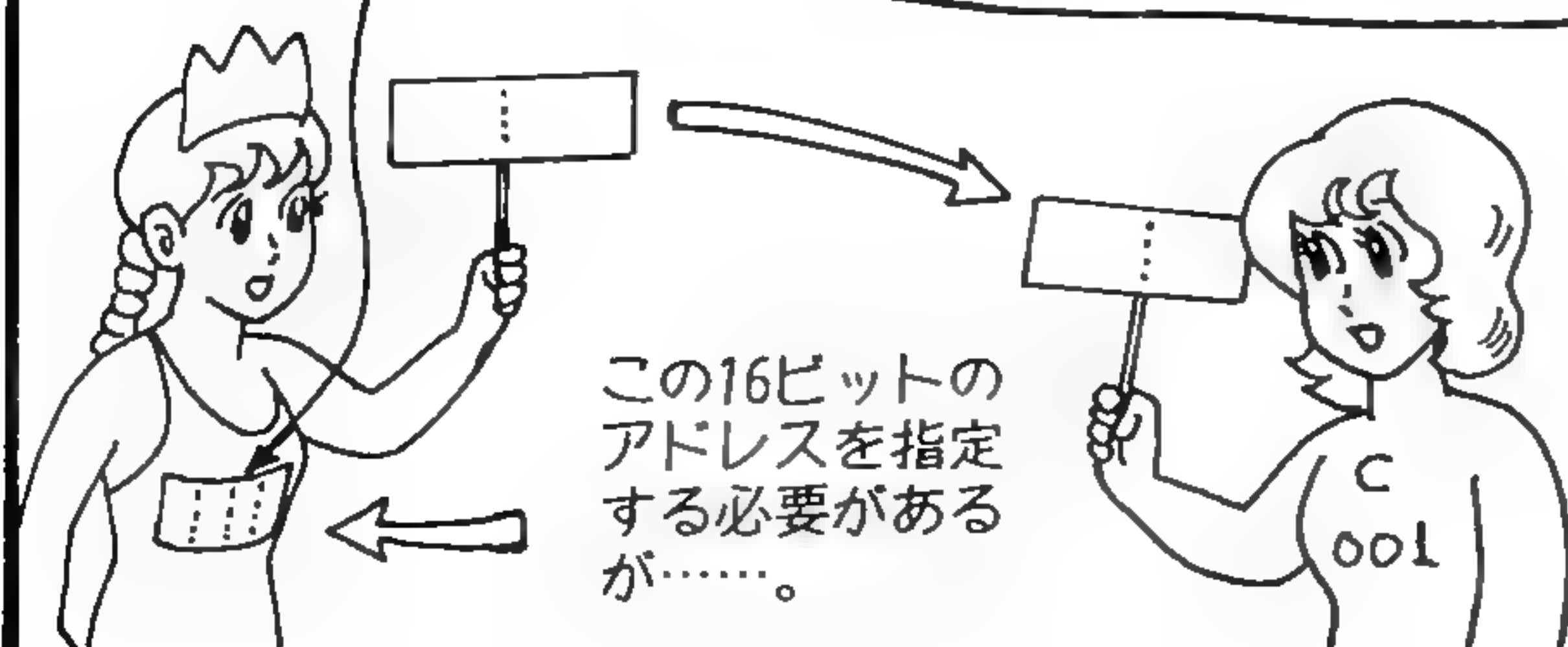
次はRAMちゃんですが、RAMちゃんはデータを送ることも受け取ることもできます。



まずROMちゃんですが、ROMちゃんはデータを送ることしかできません。ROMちゃんに書き込む命令を使っても無視されます。



たとえば外のメモリからレジスタにデータを入りたい時はアドレス〇〇〇〇番と指定する必要があります。



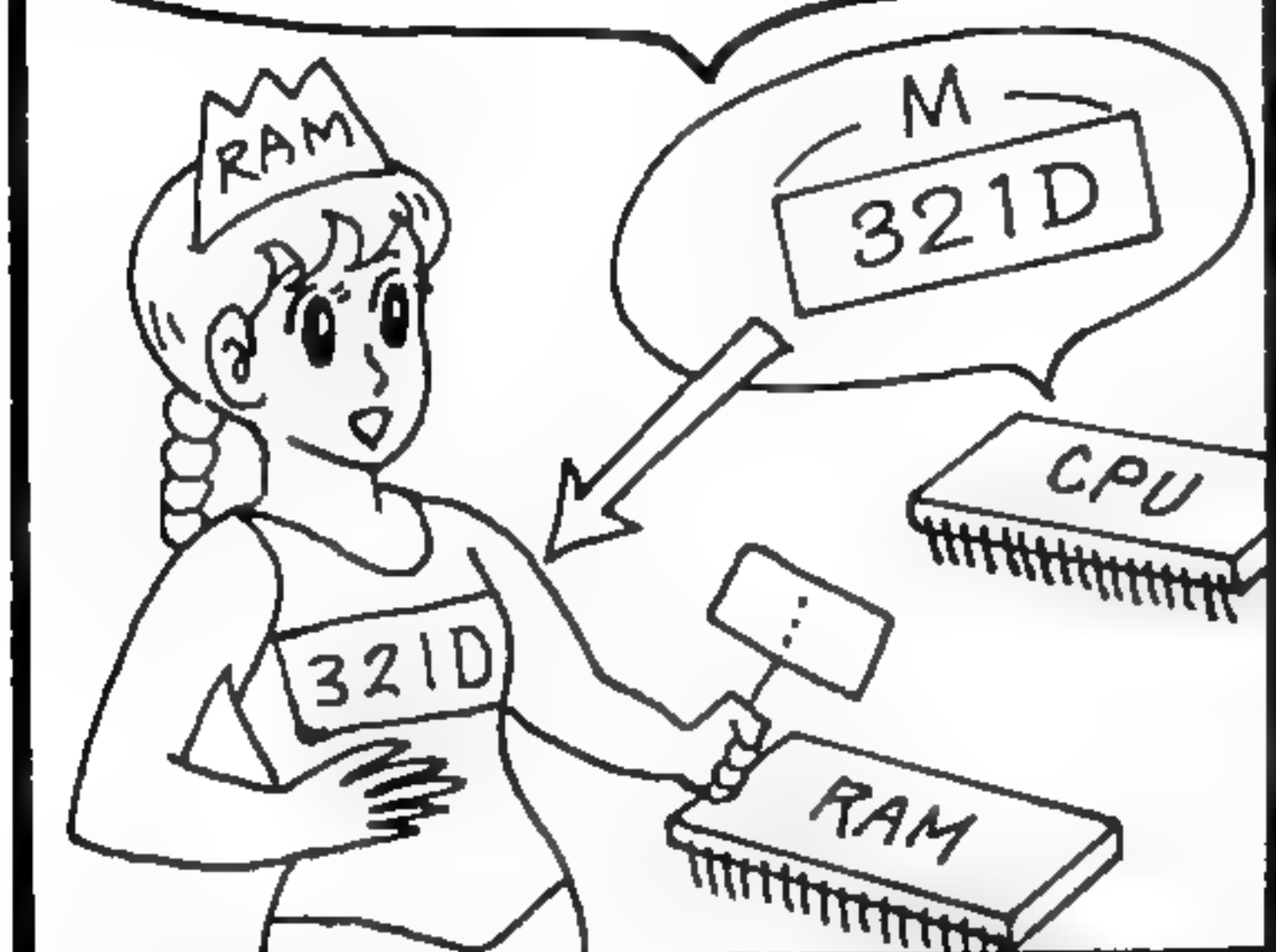
この16ビットのアドレスを指定する必要があるが……。

レジスタはA、B、C、D、E、H、Lだけでしたがメモリは0000~FFFFまで、なんとRAMちゃん65536人分使えるのです(ただし、メモリが実装されていたらのお話ですが……)。



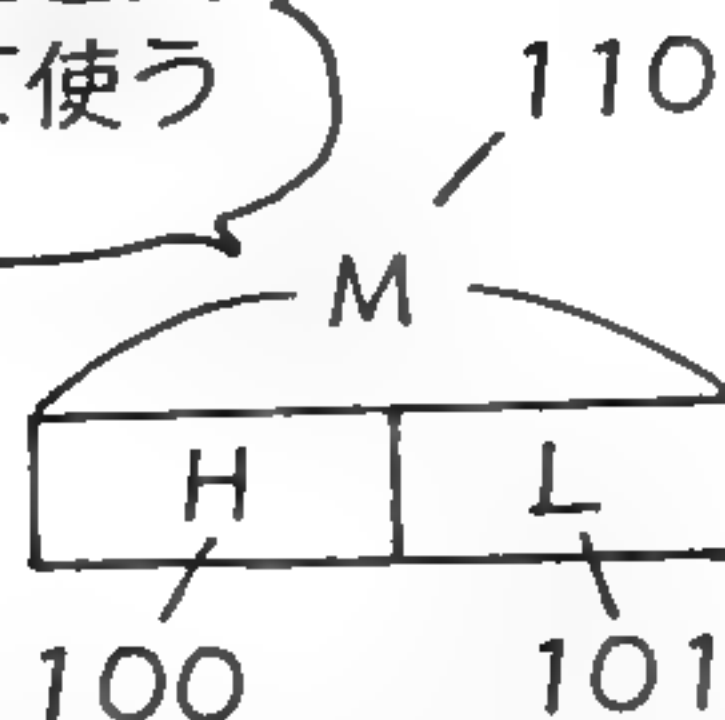


このMつまりHLレジスタの内容がアドレスを示すのです。

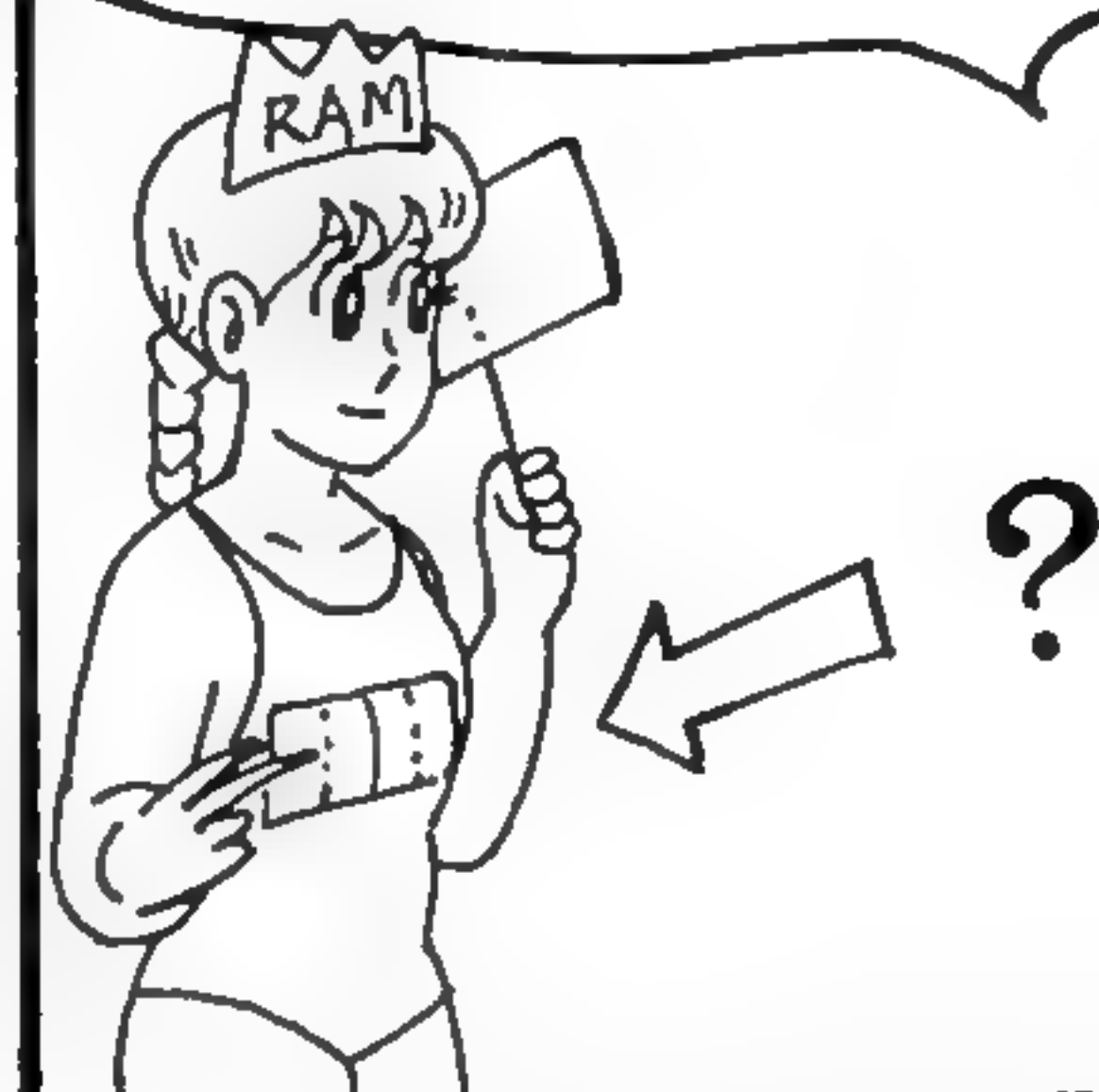


さきほどお話した、HLレジスタペアにMという名前をつけてこのコードを110としたのです。

HとLをMとして使う



実はこの16ビットのアドレス指定はとてもうまく考えられています。



ではMOV L, M, Cのマシン語を作ってみましょう。簡単ですね。71 (ナナイチ) です。

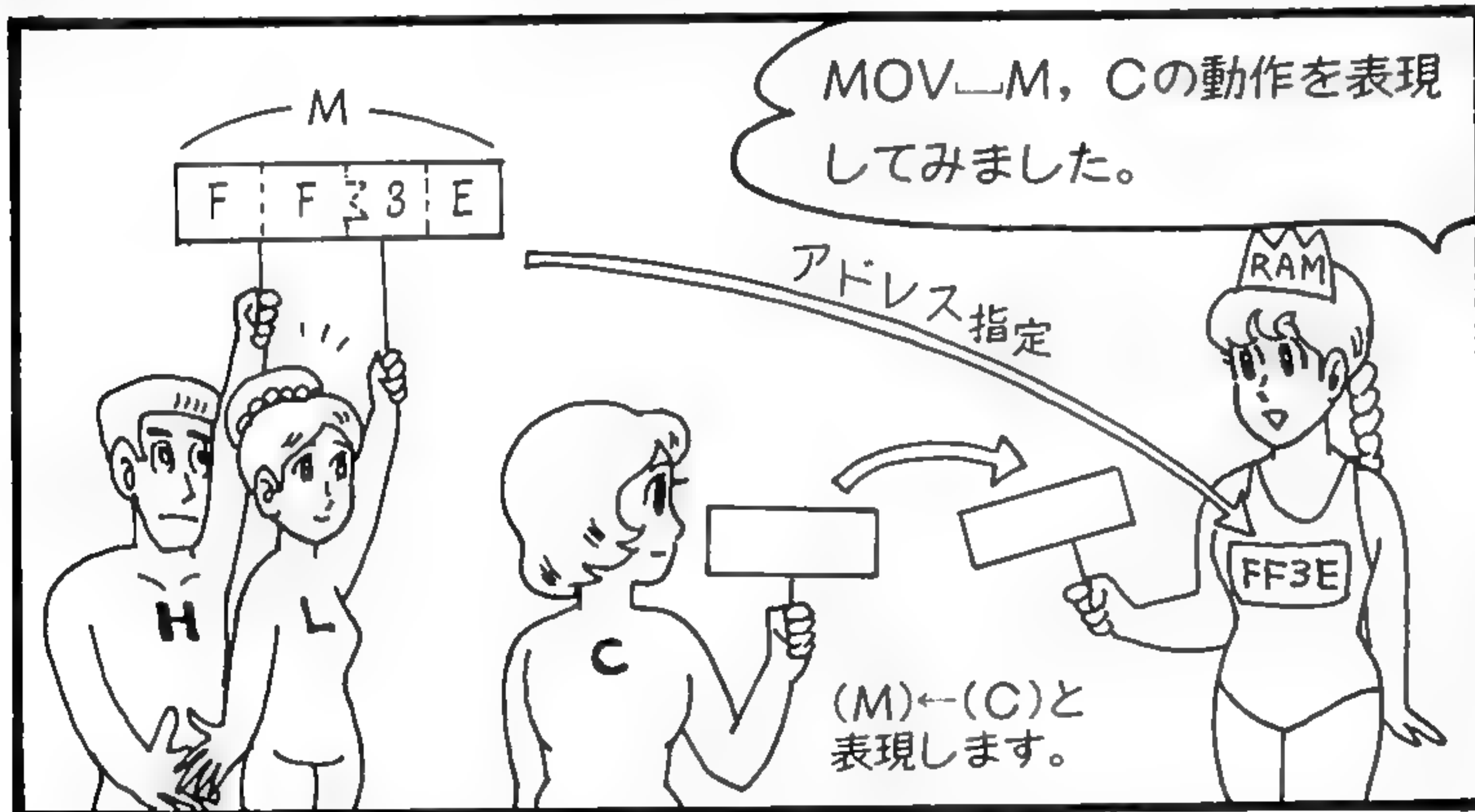
7 1  
01110001  
M C

レジスタからメモリへのデータ転送のマシン語はこうなります。

01110SSS  
M

メモリからレジスタへデータを送る時のマシン語はこうなり、

01SSS110  
M



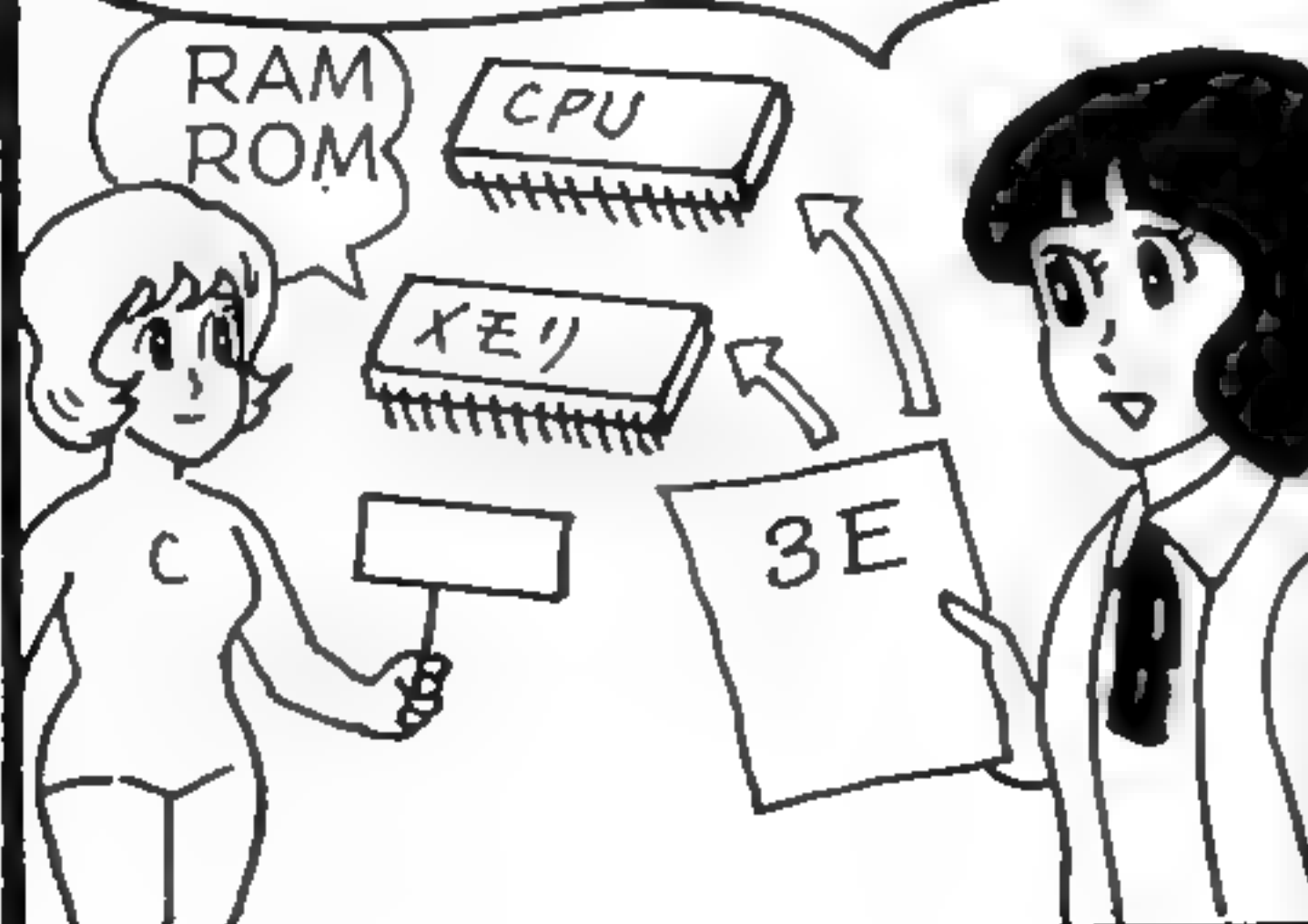
MOV L, C, Mはどうでしょうか。4E (ヨンイー) となりました。

4 E  
01001110  
C M

まず、1バイトのデータをレジスタに入れるためのマシン語です。DDDはA、B、C、D、E、H、L、Mのコードを入れますがMのときはレジスタでなくメモリにデータが入ることになります。

00DDD110

次は1バイトや2バイトのデータをソフト的に入れたい時のマシン語の作り方をお話しましょう。



MOV A, A というマシン語はあるのですが、MOV L, M, Mはありません。そのマシン語は76 (ナナロク) となりますがこのコードはHLT (ホルト) といってCPUは停止してしまいます。

7 6  
01110110  
M M

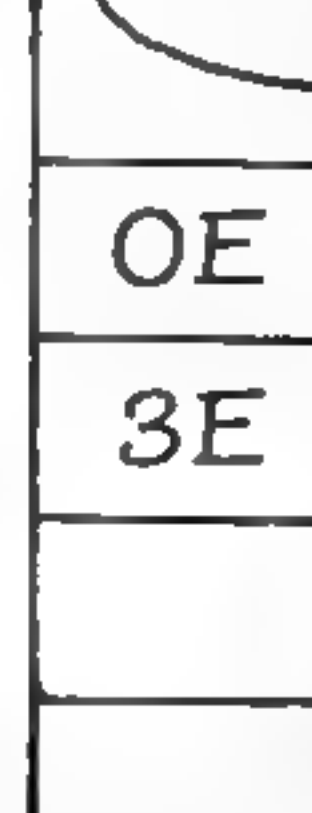


ここでひとつの疑問が生じます。このOEと3Eの区別をどうつけるのかということです。つまりこの3EはCPUに動作をさせたいマシン語ではないのです。

3Eってマシン語もあるわけでしょう？

このマシン語はOE3Eと2バイトになってしまいました。

MVI ← C, 3E



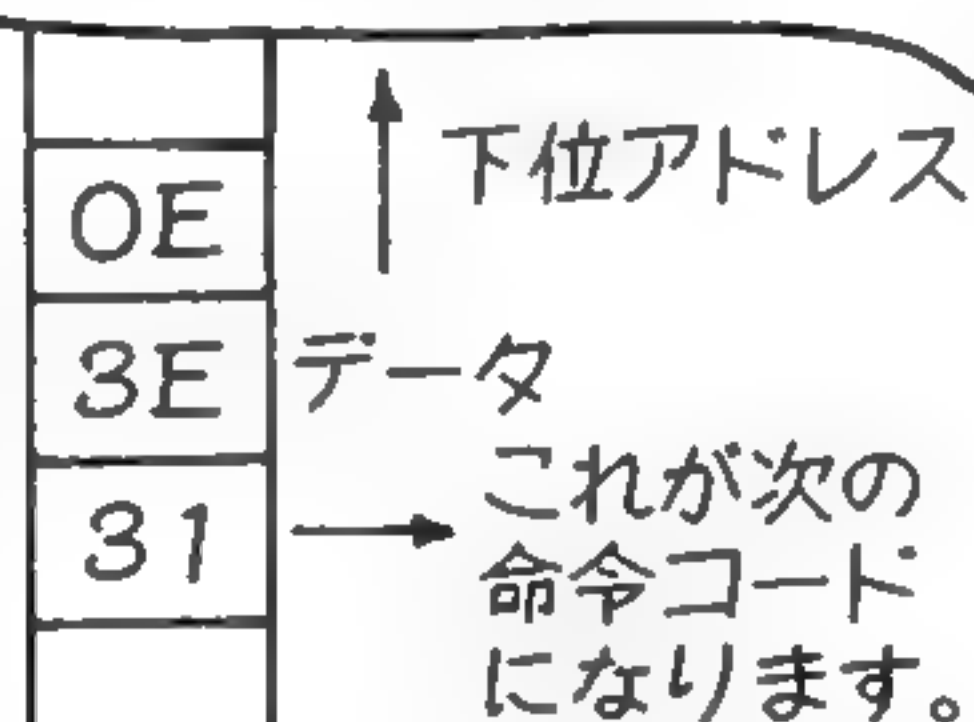
ニーモニックはMVIに変わります。MVI ← C, 3Eのマシン語はOE、その後にデータの3Eをつけます。

O E  
00001110  
C

つまり、モニタで使っていたOE3E4E0078...といったものの中には、CPUに動作させるためのコードと、そうでない定数があることがわかります。

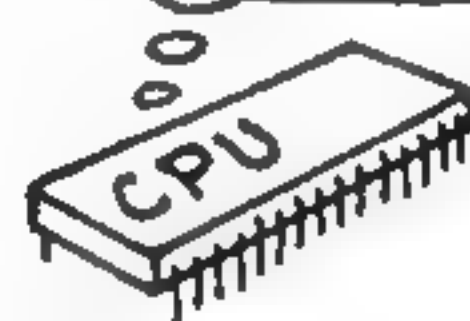
これを命令コードとして読めば  
↓これがデータとして読まれる  
31C8203E0106

つまりすぐあとの1バイトである3Eは定数として読み込まれ、次にCPUがマシン語として読むのは定数のあとの次の1バイトです。

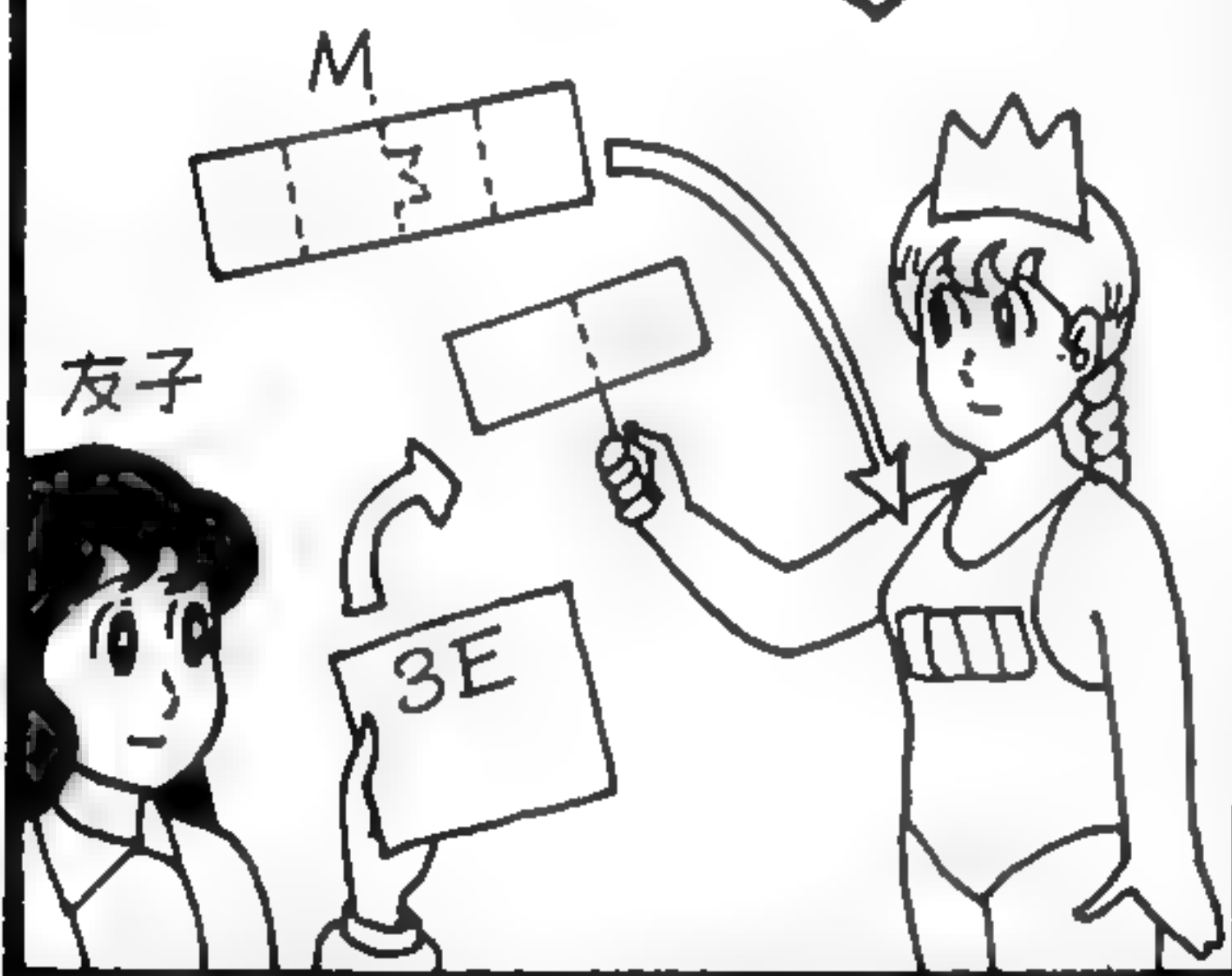


ところが、そこはうまくできています。CPUはOEを読むと次の1バイトが定数であると判断するようになっているのです。

おや、OEが来た。ではもう1バイト読み込もう。



MVI ← M, 3Eの動作です。



次にMVI ← M, 3Eのオペコードをつくってみましょう。36 (サンロク) となりました。

3 6  
001101103E  
M

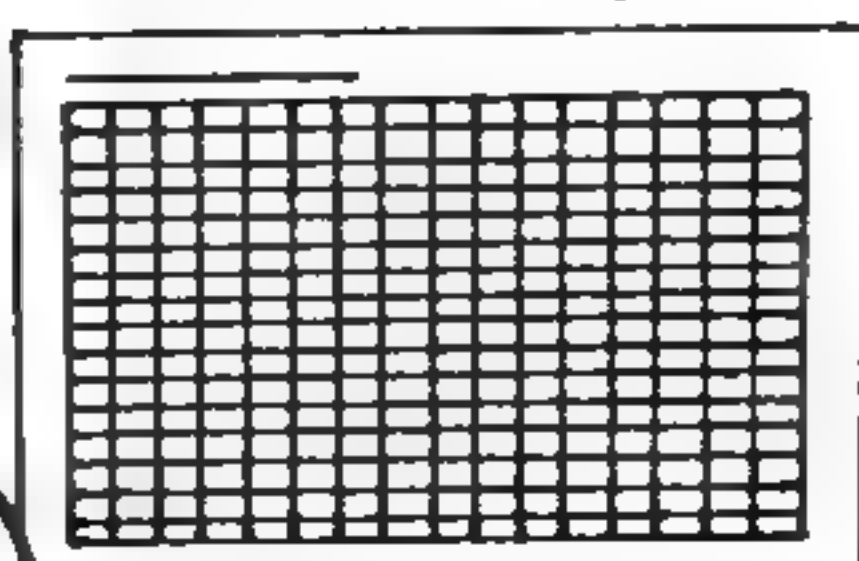
今までマシン語と言ってきましたが定数と区別するためオペコード (オペレーションコード) と言い換えることにしましょう。

MVI ← D, FF  
16FF  
↑ ↑  
オペコード データ

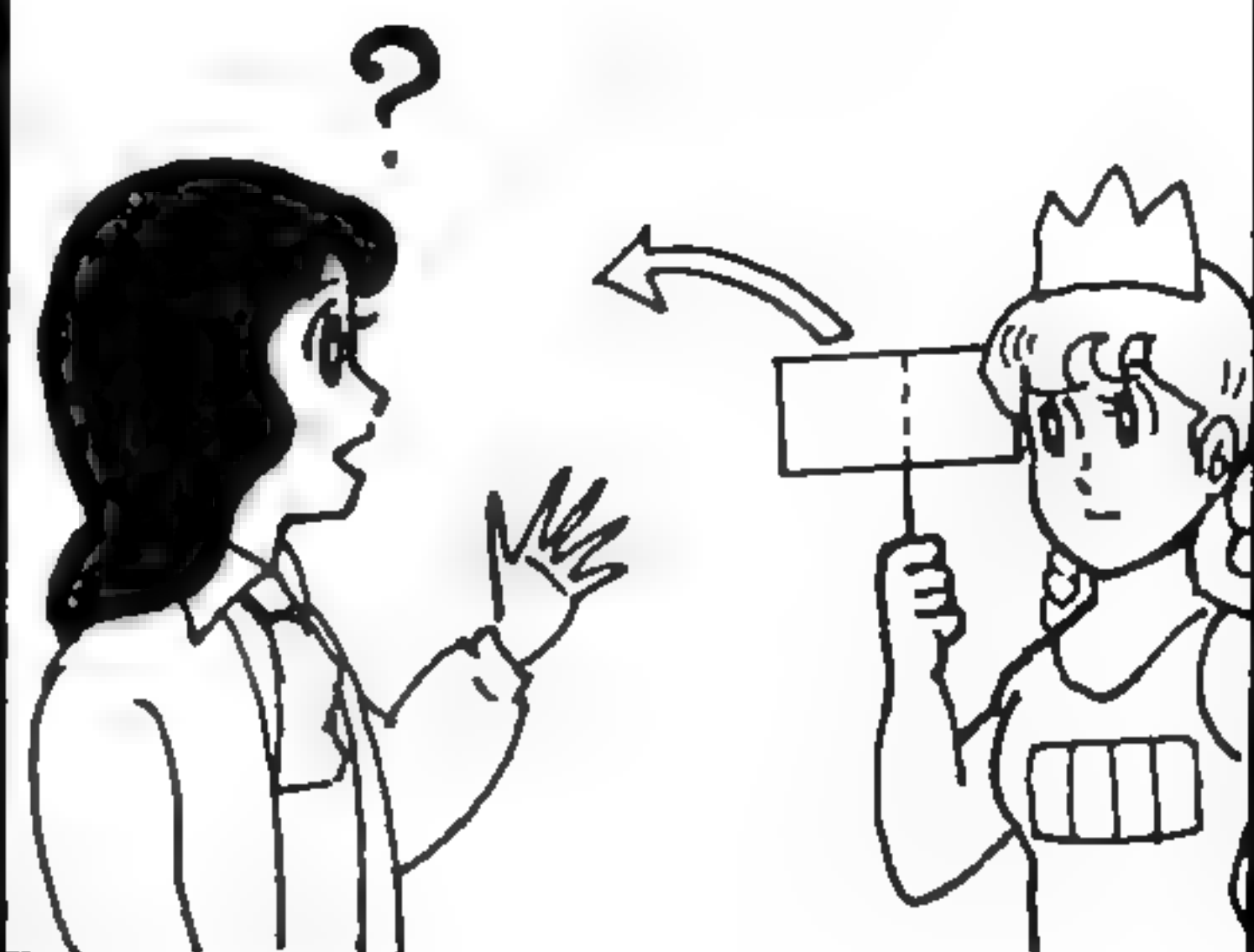
なおこれは8080用にインテル社が考えたニーモニックです。この表はアセンブラ記述形式で書いてありますのでとても役に立つものです。

MOV、MVI、LXI  
などが出てきたらこれはインテルのニーモニックということね。

次のページにニーモニックとオペコードの対照表を示します。今作ったオペコードがあるか確かめてみてください。



MVI ← 3E, Mはありません。





8080CPUのニーモニック-OPコード割当表

(Z80)はZ80CPUのOPコードに使われます。  
D8は1バイトのデータ、D16は2バイトのデータを  
示します。Addrはアドレス2バイトです。

「はスペースを示します。  
アセンブラを使用する時便利な  
ようにMOVLD,Bのようにオ  
ペランド部を含めて書いています。

これが8080の  
全命令です。



表1-7 8080⑧

上桁 下桁	0.	1.	2.	3.	4.	5.	6.	7.	8.	9.	A.	B.	C.	D.	E.	F.
・ 0	NOP	NOP (Z80)	NOP (Z80)	NOP (Z80)	MOV B,B	MOV D,B	MOV H,B	MOV M,B	ADD B	SUB B	ANA B	ORA B	RNZ	RNC	RPO	RP
・ 1	LXI B, D16	LXI D, D16	LXI H, D16	LXI SP, D16	MOV B,C	MOV D,C	MOV H,C	MOV M,C	ADD C	SUB C	ANA C	ORA C	POP B	POP D	POP H	POP PSW
・ 2	STAX B	STAX D	SHLD, Addr	STA, Addr	MOV B,D	MOV D,D	MOV H,D	MOV M,D	ADD D	SUB D	ANA D	ORA D	JNZ, Addr	JNC, Addr	JPO, Addr	JP, Addr
・ 3	INX B	INX D	INX H	INX SP	MOV B,E	MOV D,E	MOV H,E	MOV M,E	ADD E	SUB E	ANA E	ORA E	JMP, Addr	OUT D8	XTHL	DI
・ 4	INR B	INR D	INR H	INR M	MOV B,H	MOV D,H	MOV H,H	MOV M,H	ADD H	SUB H	ANA H	ORA H	CNZ, Addr	CNC, Addr	CPO, Addr	CP, Addr
・ 5	DCR B	DCR D	DCR H	DCR M	MOV B,L	MOV D,L	MOV H,L	MOV M,L	ADD L	SUB L	ANA L	ORA L	PUSH B	PUSH D	PUSH H	PUSH PSW
・ 6	MVI B, D8	MVI D, D8	MVI H, D8	MVI M, D8	MOV B,M	MOV D,M	MOV H,M	HLT	ADD M	SUB M	ANA M	ORA M	ADI D8	SUI D8	ANI D8	ORI D8
・ 7	RLC	RAL	DAA	STC	MOV B,A	MOV D,A	MOV H,A	MOV M,A	ADD A	SUB A	ANA A	ORA A	RST 0 この「0」は11-15の「7」に入れる	RST 2	RST 4	RST 6
・ 8	NOP (Z80)	NOP (Z80)	NOP (Z80)	NOP (Z80)	MOV C,B	MOV E,B	MOV L,B	MOV A,B	ADC B	SBB B	XRA B	CP B	RZ	RC	RPE	RM
・ 9	DAD B	DAD D	DAD H	DAD SP	MOV C,C	MOV E,C	MOV L,C	MOV A,C	ADC C	SBB C	XRA C	CP C	RET	RET (Z80)	PCHL	SPHL
・ A	LDAX B	LDAX D	LHLD, Addr	LDA, Addr	MOV C,D	MOV E,D	MOV L,D	MOV A,D	ADC D	SBB D	XRA D	CP D	JZ	JC, Addr	JPE, Addr	JM, Addr
・ B	DCX B	DCX D	DCX H	DCX SP	MOV C,E	MOV E,E	MOV L,E	MOV A,E	ADC E	SBB E	XRA E	CP E	JMP (Z80)	IN D8	XCHG	EI
・ C	INR C	INR E	INR L	INR A	MOV C,H	MOV E,H	MOV L,H	MOV A,H	ADC H	SBB H	XRA H	CP H	CZ, Addr	CC, Addr	CPE, Addr	CM, Addr
・ D	DCR C	DCR E	DCR L	DCR A	MOV C,L	MOV E,L	MOV L,L	MOV A,L	ADC L	SBB L	XRA L	CP L	CALL, Addr	CALL (Z80)	CALL (Z80)	CALL (Z80)
・ E	MVI C, D8	MVI E, D8	MVI L, D8	MVI A, D8	MOV C,M	MOV E,M	MOV L,M	MOV A,M	ADC M	SBB M	XRA M	CP M	ACI D8	SBI D8	XRI D8	CPI D8
・ F	RRC	RAR	CMA	CMC	MOV C,A	MOV E,A	MOV L,A	MOV A,A	ADC A	SBB A	XRA A	CP A	RST 1	RST 3	RST 5	RST 7



# ⑦マシン語 こんな命令もある

プログラム  
8E →  
データ  
(M) 97 →

RAM → CPU

おい次の命令は8Eだ。  
(A) ← (A) + (M) + CARRY  
をやるぞ。

???

Aレジスタ  
HLレジスタペア  
キャリーフラグ  
集まれ! あっそうだ。  
データ(M)は外のRAM  
からもってこなきゃ。



D7 D6 D5 D4 D3 D2 D1 D0

FLAGS OF 8080A  
無視できないよ。このウーマンパワー

D7 サイン  
D6 ゼロ  
D4 補助キャリー  
D2 パリティ  
D0 キャリー

ADC M  
8E (これは1バイト命令だからONEパターン)

ビットに直してたし算するのがコツです。答はもう一度16進に。

与式  
 $CY + (A) + (M)$   
 $= 1 + 28 + 97$   
≠ CO

16進の計算ができる人には、この答が合っていることがわかりますね。

いやんなっちゃうんだよな。10進補正しないと答が合わないんだ。

(M) ← (H)(L)  
(M) メモリのアドレス指定

メモリ RAM  
2003 97

ANI 38  
〈例〉  
E6 38 ⇒ 00111100

この命令は特定のビットを調べたり、マスクしたりするのに用いられる。8bitのANDがソフトで実現できるわけ

このAレジスタの内容は、たとえばインポートポートからあらかじめ読み込まれたもの

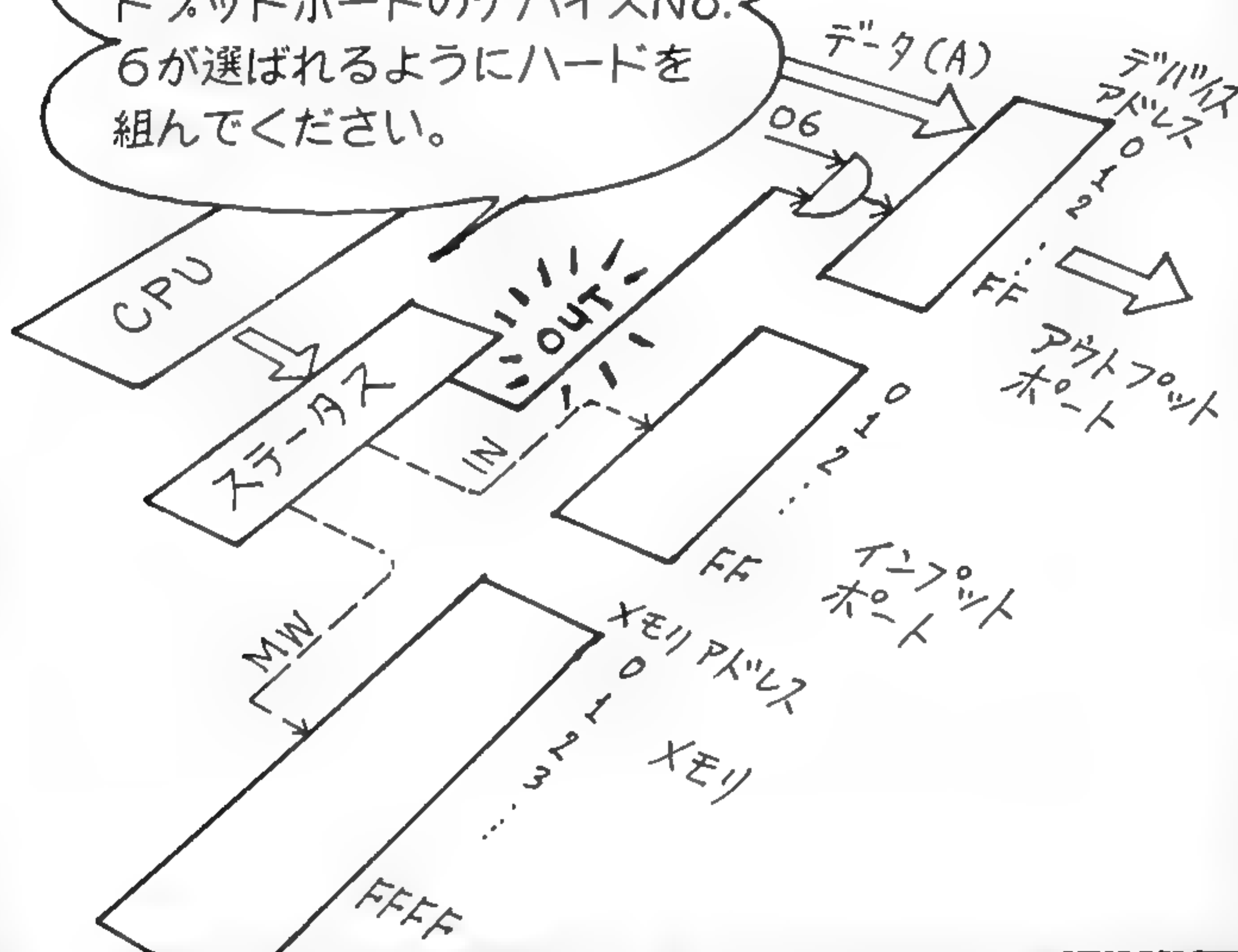
結果はAレジスタに戻される。

フラグ  
“ > > > > ”

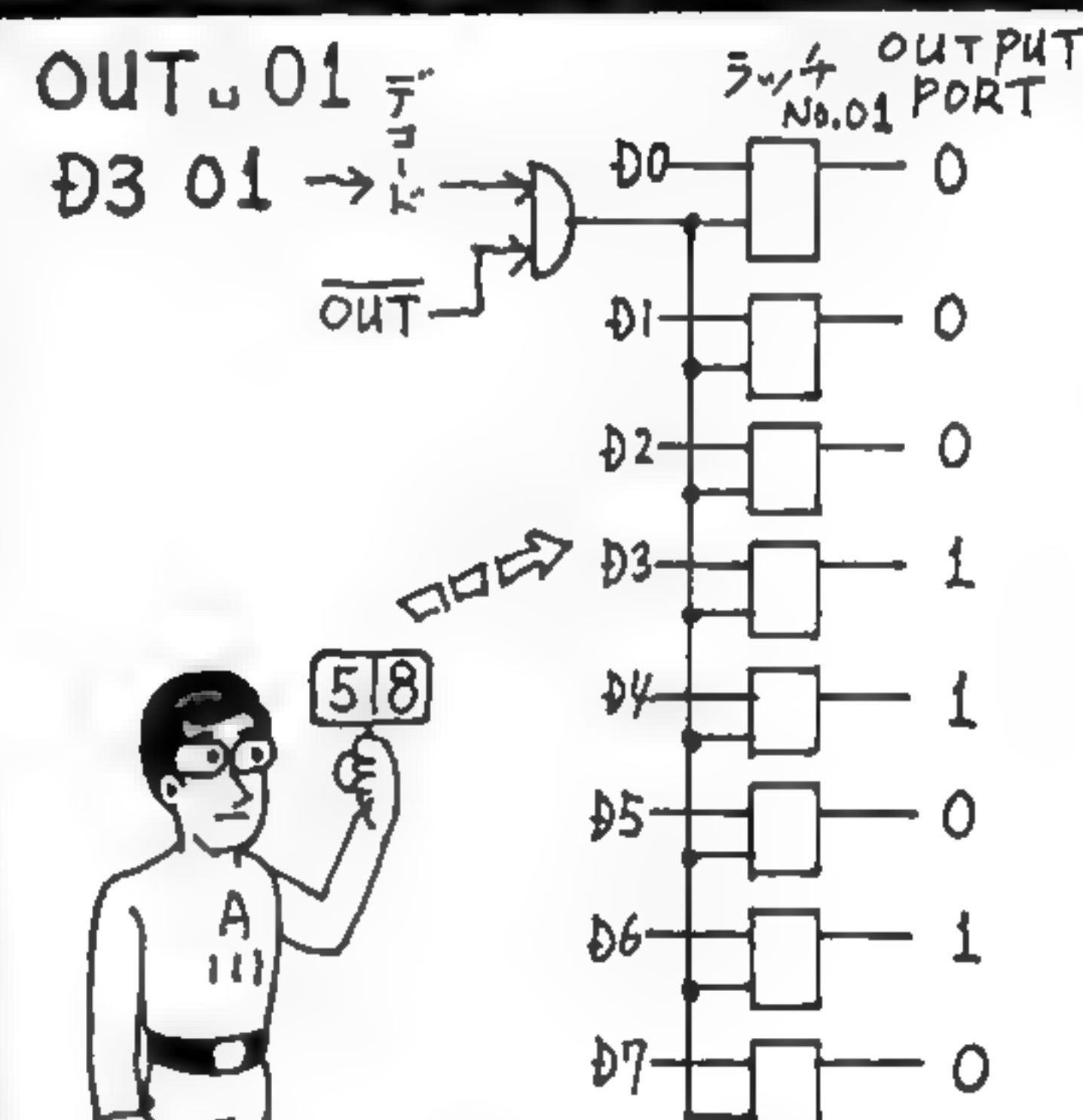
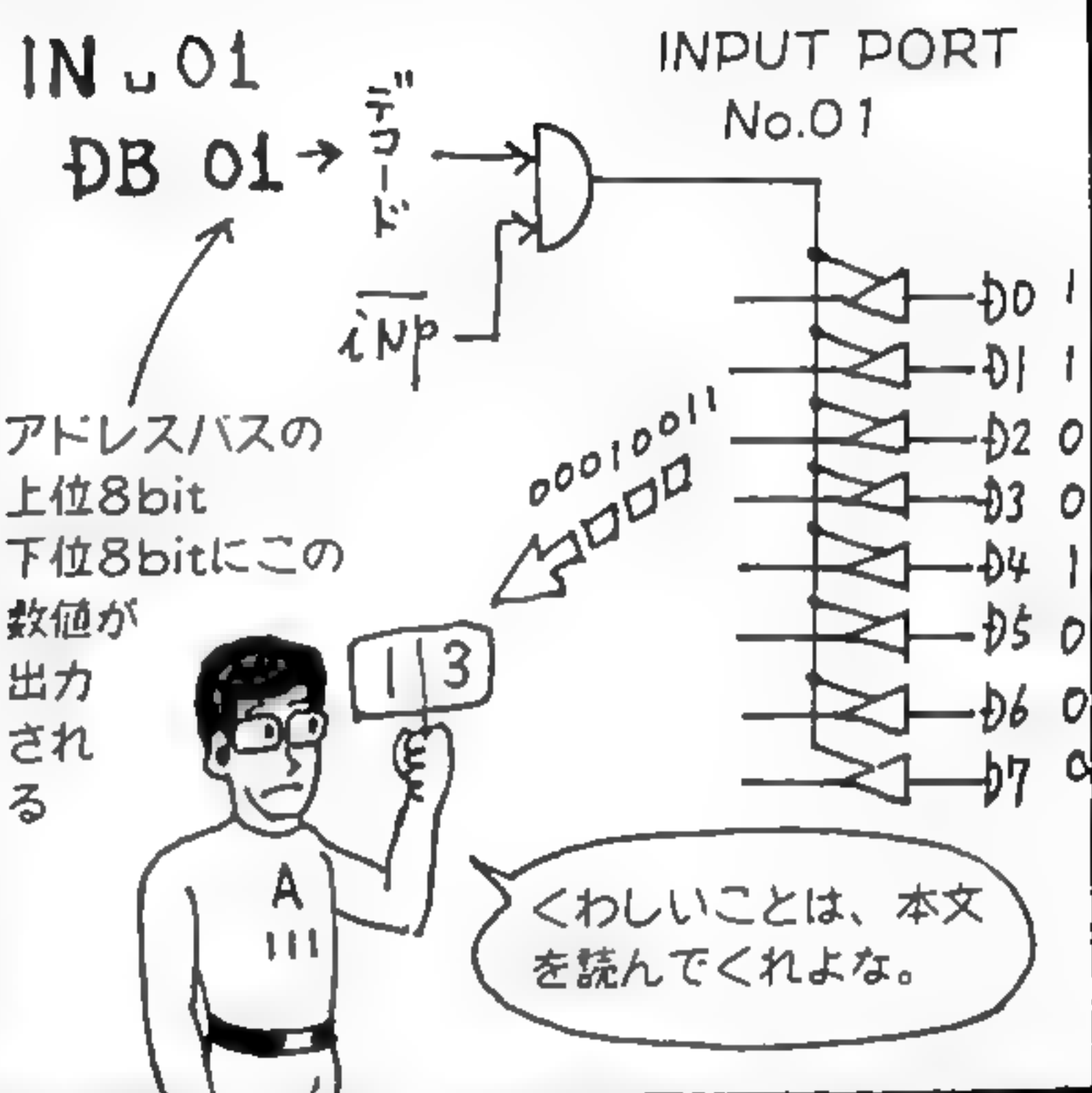
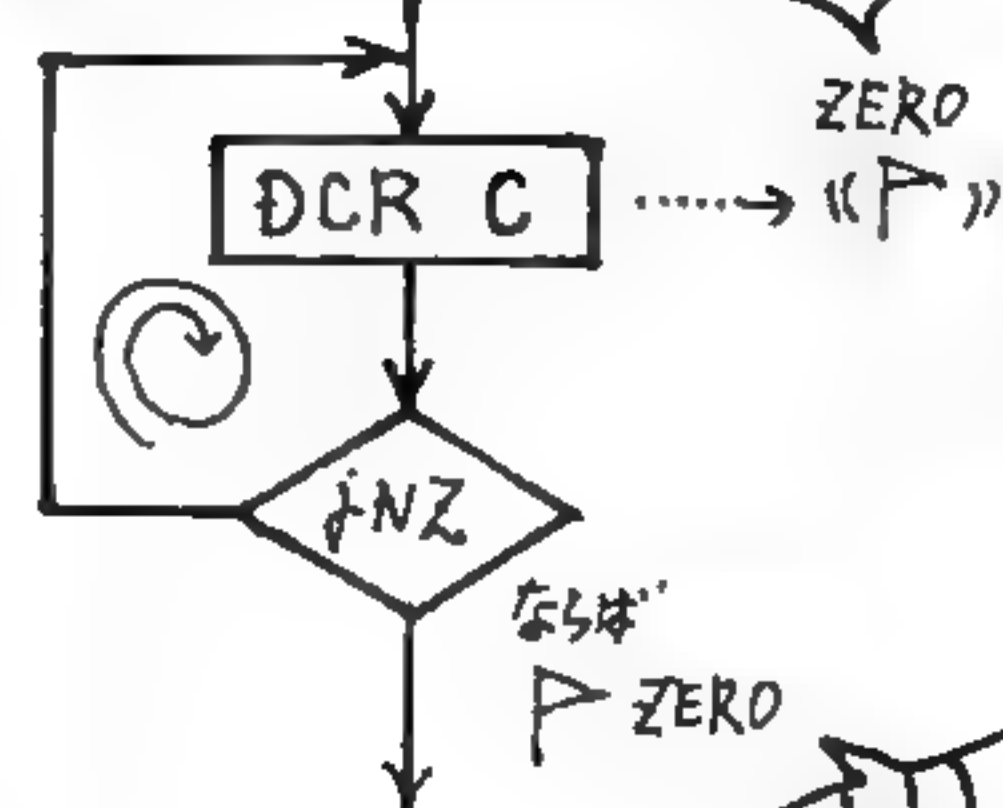


メモリの他にインポートポートとアウトポートポートがついているのが8080系CPUの特徴のひとつです。それぞれ256バイト分のデバイスNo.を指定できます。

OUT 06を実行するとアウトポートポートのデバイスNo. 6が選ばれるようにハードを組んでください。

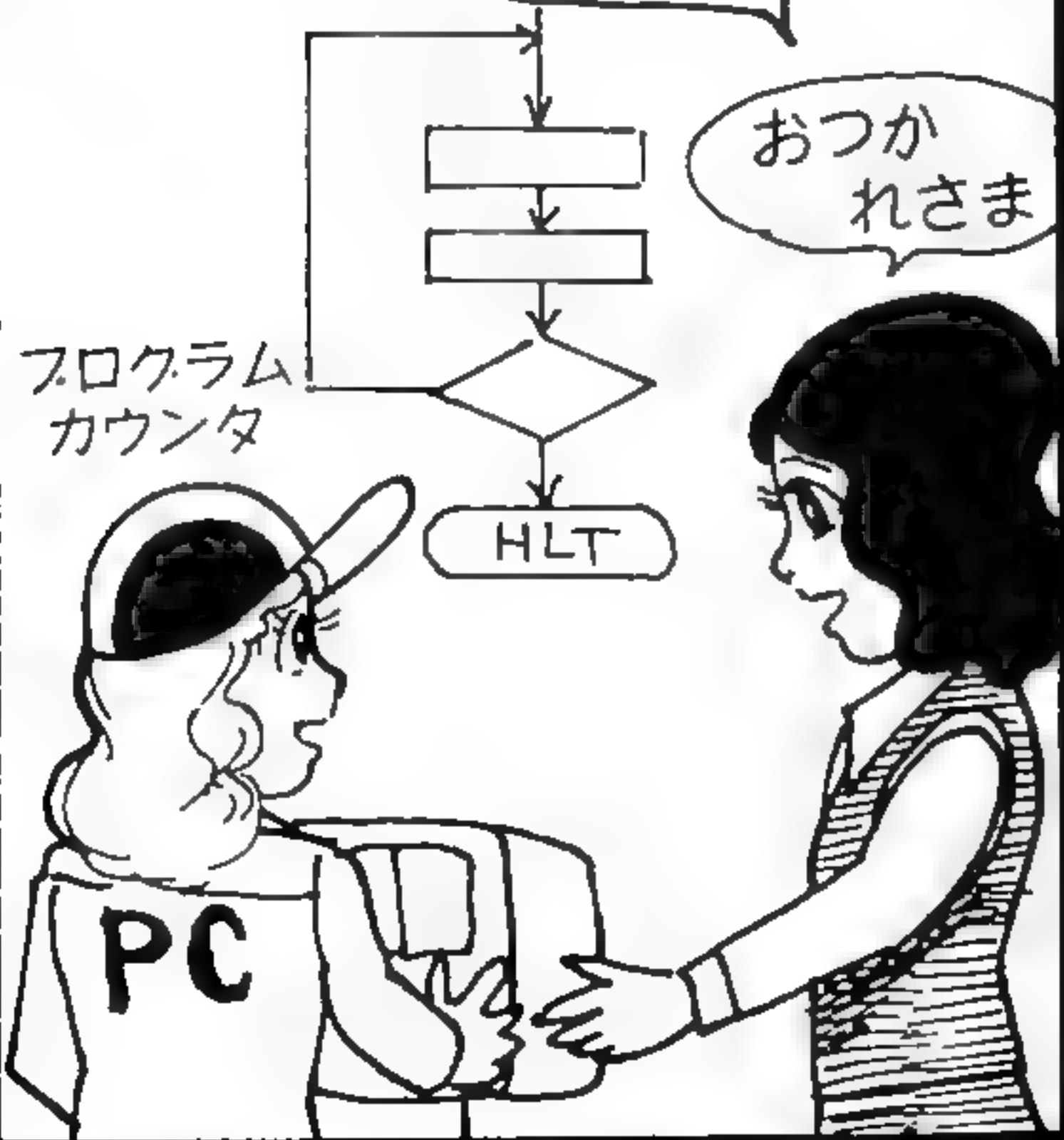


インクリメント、デクレメントという命令は本当にコンピュータ的なんです。使いこなせばこんな便利な命令はありません。



## HLT (ホルト) 76

何事も始めがあれば終わりがあります。この命令の実行後はリセット信号をハード的に入れるまでプログラムの実行は停止します。



## NOP (ノーオペレーション)

00 NOP  
01 NOP  
02 NOP  
03 NOP  
04 NOP  
05 NOP  
06 NOP  
07 NOP  
08 NOP  
09 NOP  
0A NOP  
0B NOP  
0C NOP  
0D NOP  
0E NOP  
0F NOP

この命令はなにもしないという変な命令です。

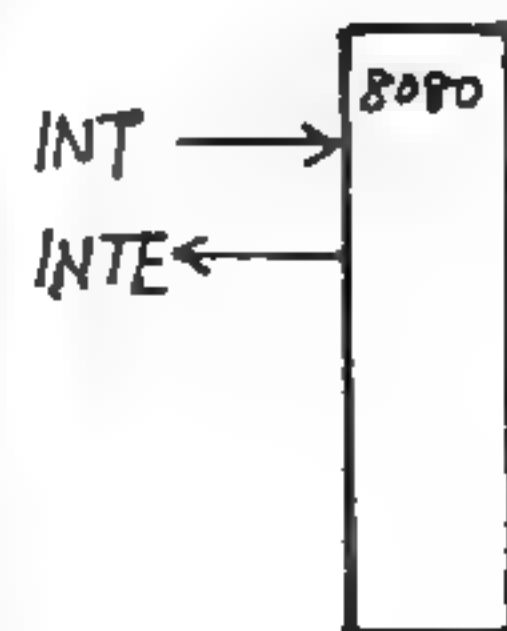
でも何もしないという命令を読み込むために4ステート分の時間を消費しました。これをあれこれ応用するんです。



## EI FB

この命令を実行させないと外部からの割り込みができません。

(インタラフトイネーブル)





DAA命令を使って16進数を10進数に変換する。

BCD下桁      BCD上桁

(A) 1101 1011

BCDコードではA、B、C、D、E、Fは禁止コードです。

0~9以外のコードだったら6を加えて0~9にします。

16進      BCD

(A) 1010 → 0000 (0)  
 (B) 1011 → 0001 (1)  
 (C) 1100 → 0010 (2)  
 (D) 1101 → 0011 (3)  
 (E) 1110 → 0100 (4)  
 (F) 1111 → 0101 (5)

CARRY

桁上げ分を上桁に加える

ALUの演算は8ビットの加減算のみで、答は16進数です。10進数の答がほしい時は、このDAA命令を計算命令の直後に入れておけばOKです。

DAA 27

ALUはCPUの中にあります。

(A) ← (A) これはどんなことに使うのかというと、減算を加算で行なう場合です。この命令と、インクリメント、デクレメント、それから加算を組み合わせると減算ができるのです。CMA

5  
- 6  
—  
- 1

↓ ALUに5を入力  
↓ 補数をとる  
↓ 2の補数にするため  
↓ インクリメント(+1)  
↓ 加算する  
↓ 0101 (5)  
+ 1010  
—  
↓ 1111  
↓ 桁上げがなければ、答は負数である  
↓ 負の場合は答をデクレメント(-1)  
↓ 1110  
↓ 補数をとる  
↓ 0001  
∴ 答は - 1 である。

1001 (9) ... ①  
 x) 0111 (7) ... ②  
 ———  
 1001  
 10010 ..... ①を左に1ビット回転  
 +) 100100 ..... ②を左に2ビット回転  
 ———  
 111111  
 3      F

↓  
 16進数なので10進補正のDAA命令を実行すると、  
 3 × 16 = 48  
 15 × 1 = 15  
 ———  
 63  
 63になります。

9 × 7 = 63ですね。

乗・除算は左右の回転命令と加算命令の組み合わせでできるんです。

基本的にはそうですが、実際のプログラムは長くなるので省略します。

回転命令? この変な命令にもちゃんと用途があるんです。乗除算のためにはなくてはならないものです。

RLC 07

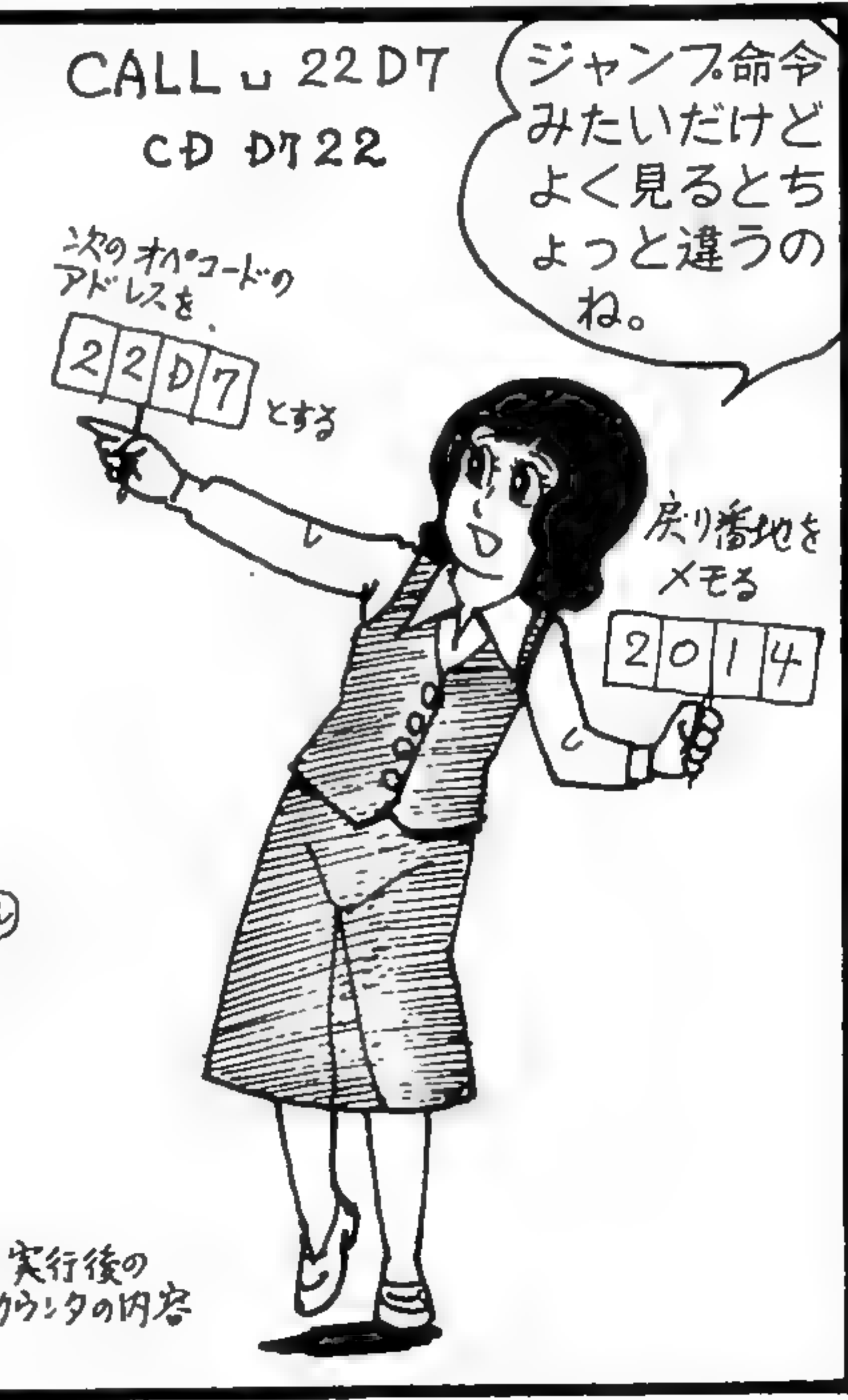
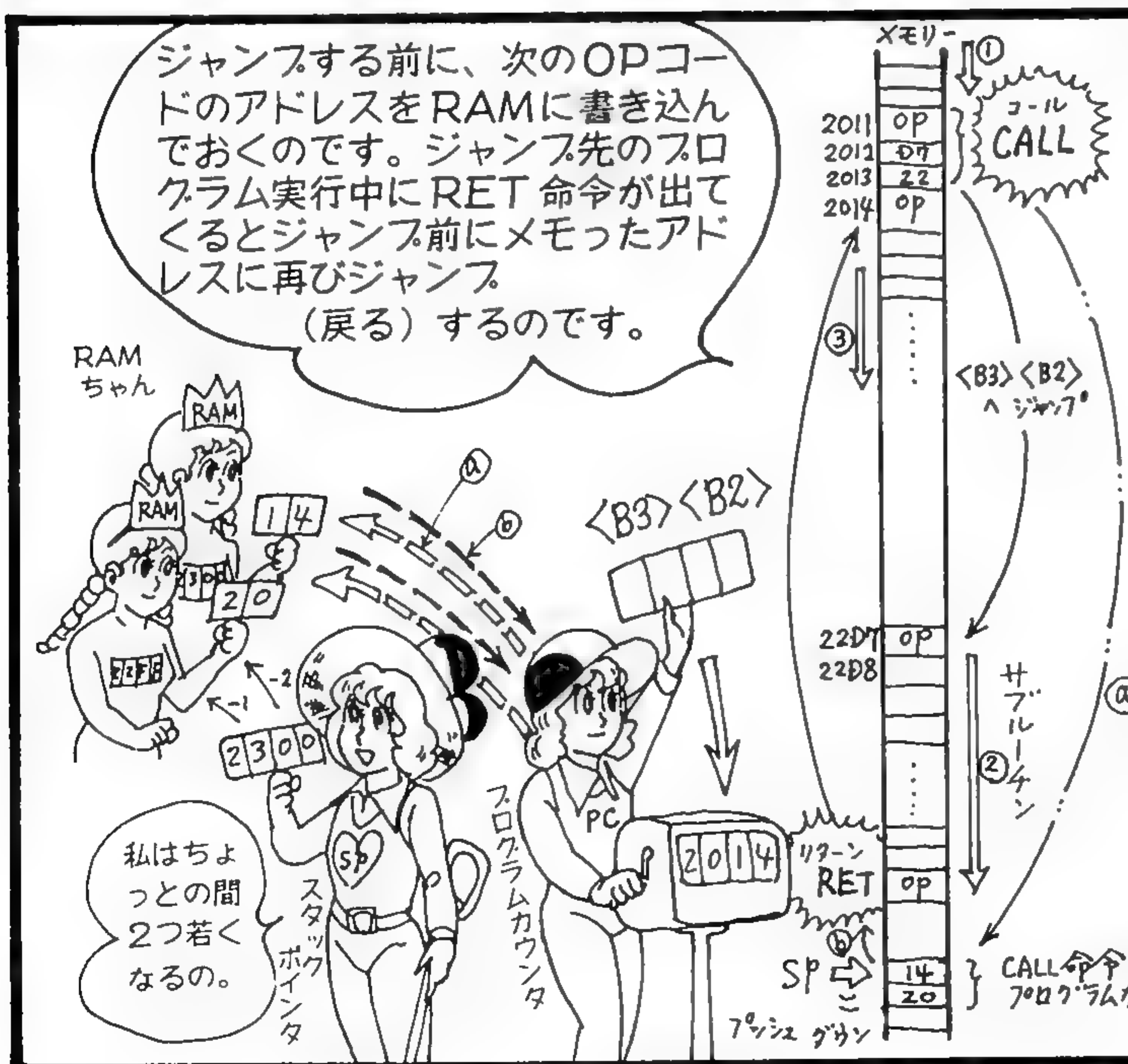
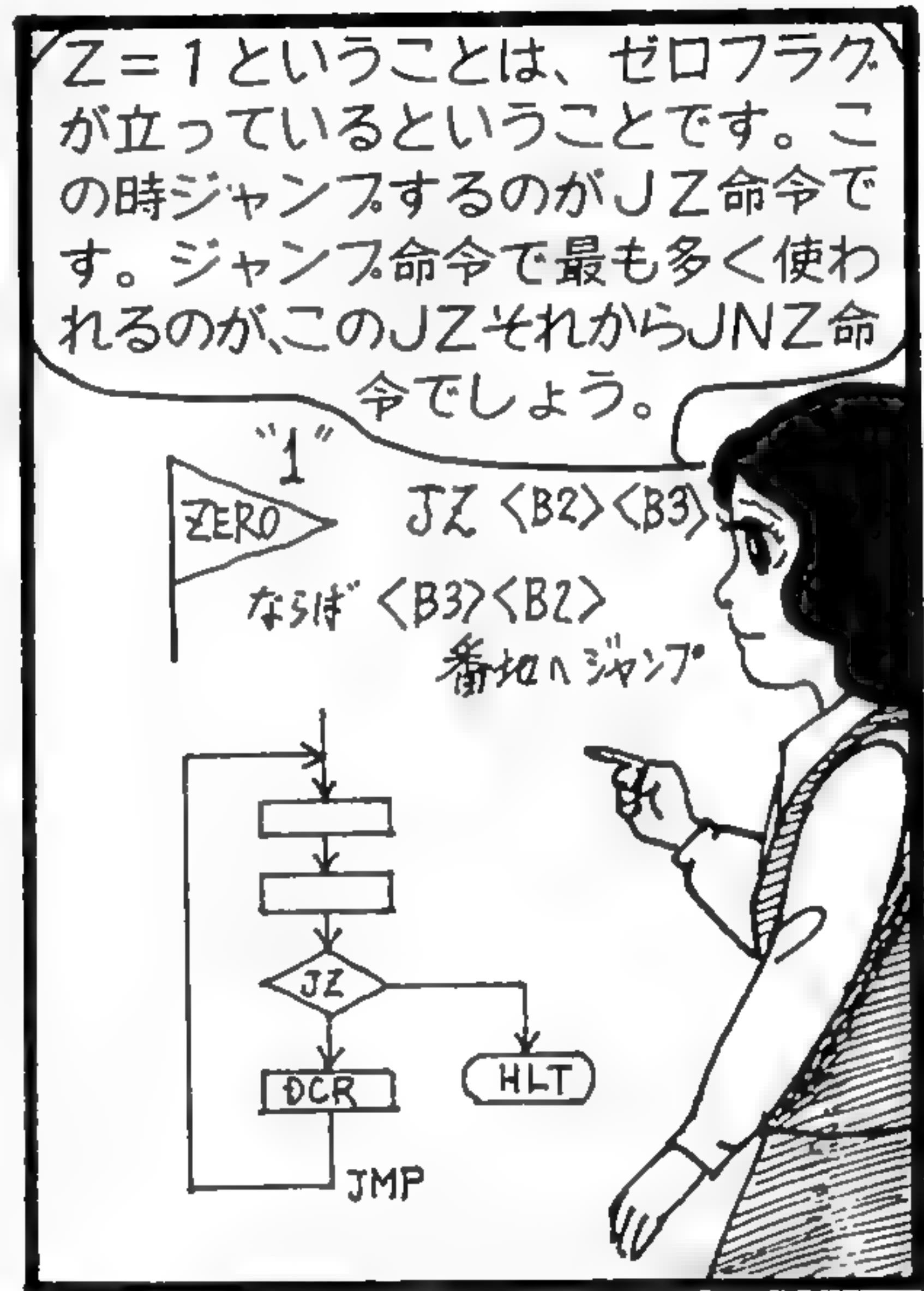
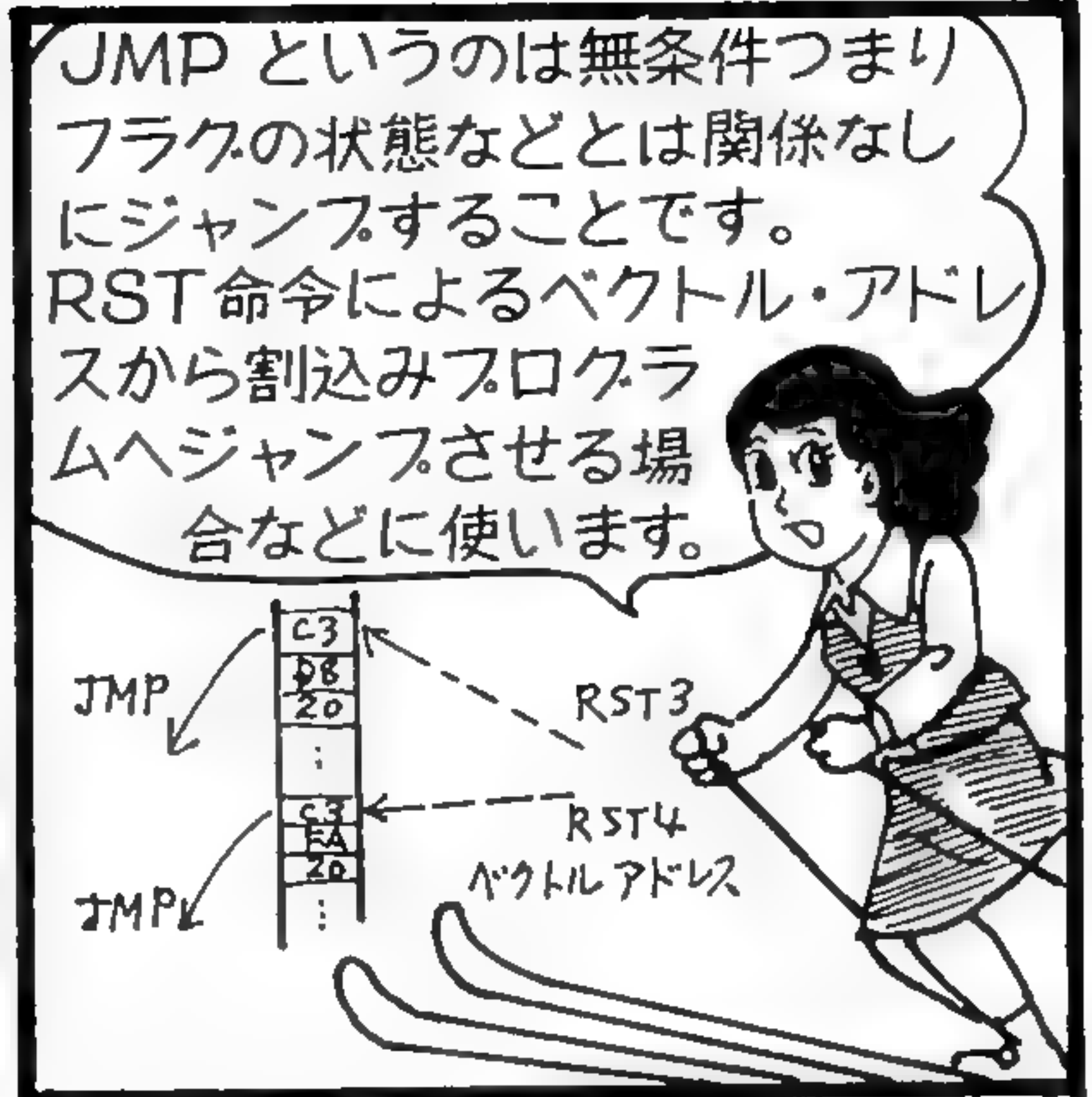
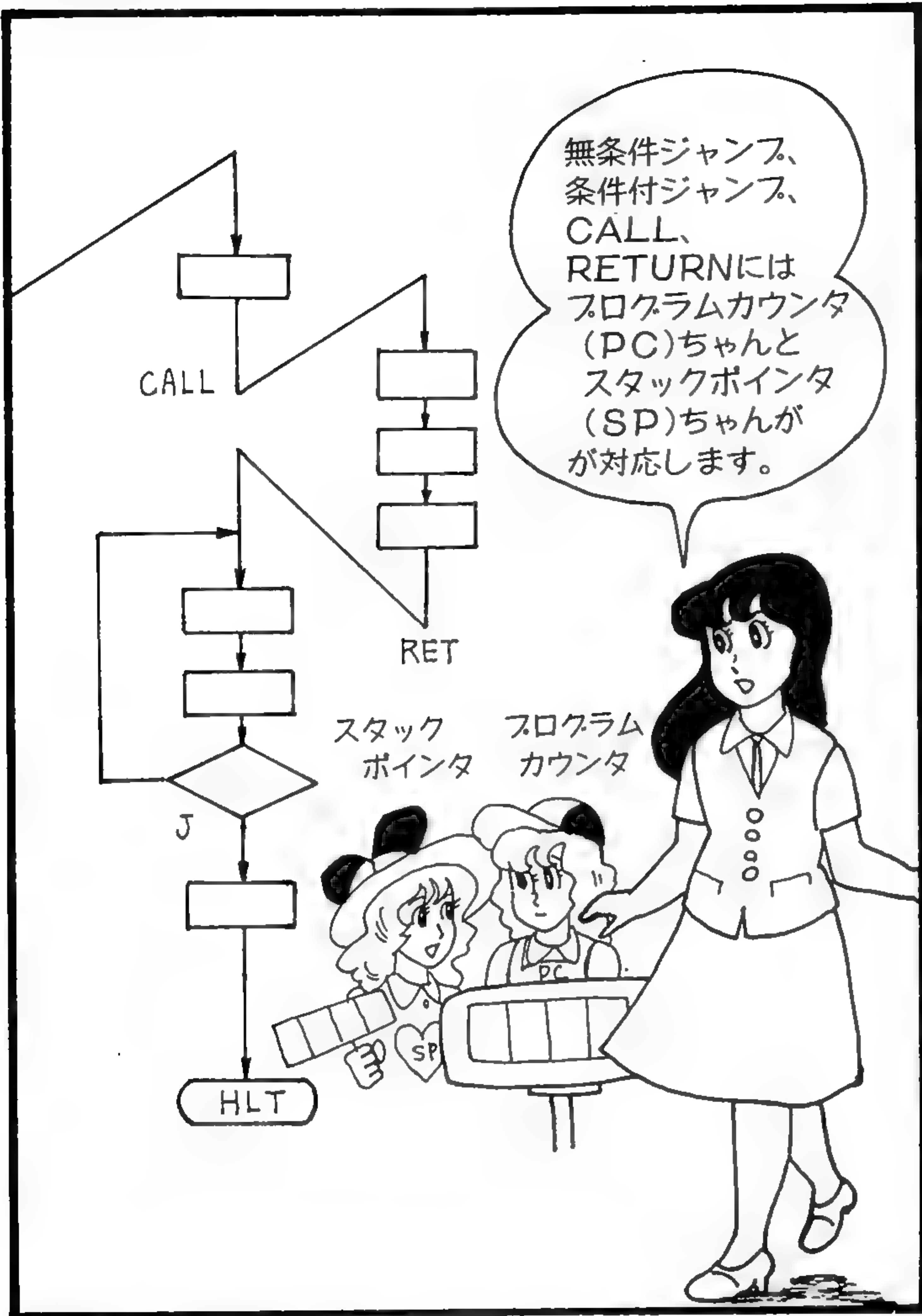
FLAGS

CF 1

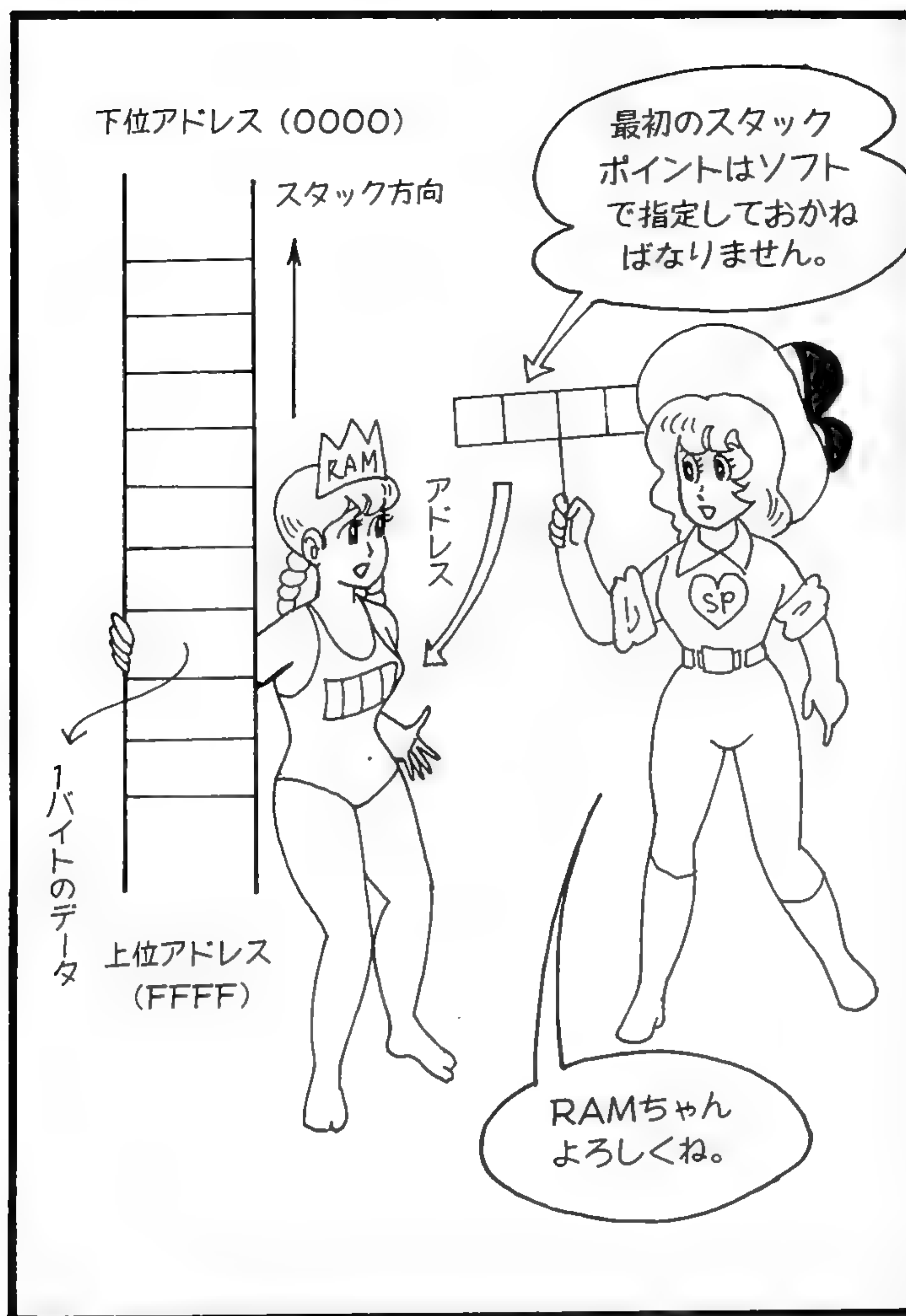
0 0 1 0 0 1 1 0  
 D7 D6 D5 D4 D3 D2 D1 D0  
 命令実行後 ↓  
 0 1 0 0 1 1 0 0

↑ (C) ← D7      ← 1ビットずらす (D0) ← (D7)









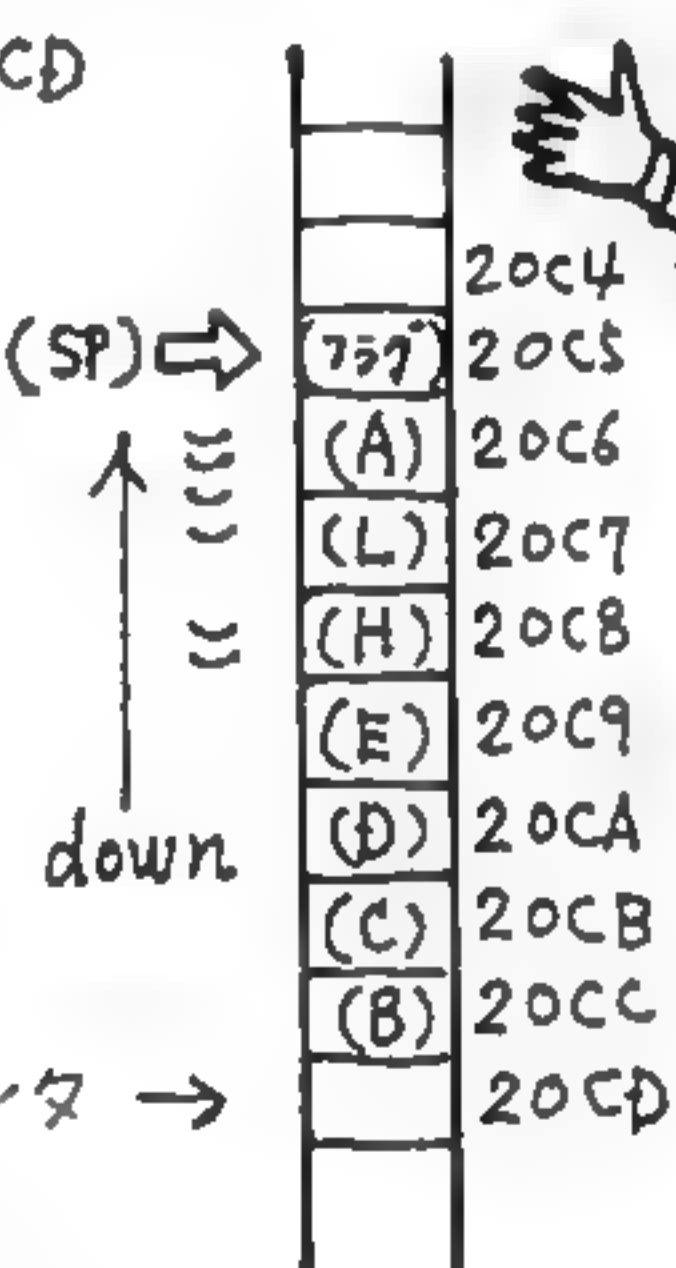
CALL命令とRET命令は必ず対になっていなければなりません。



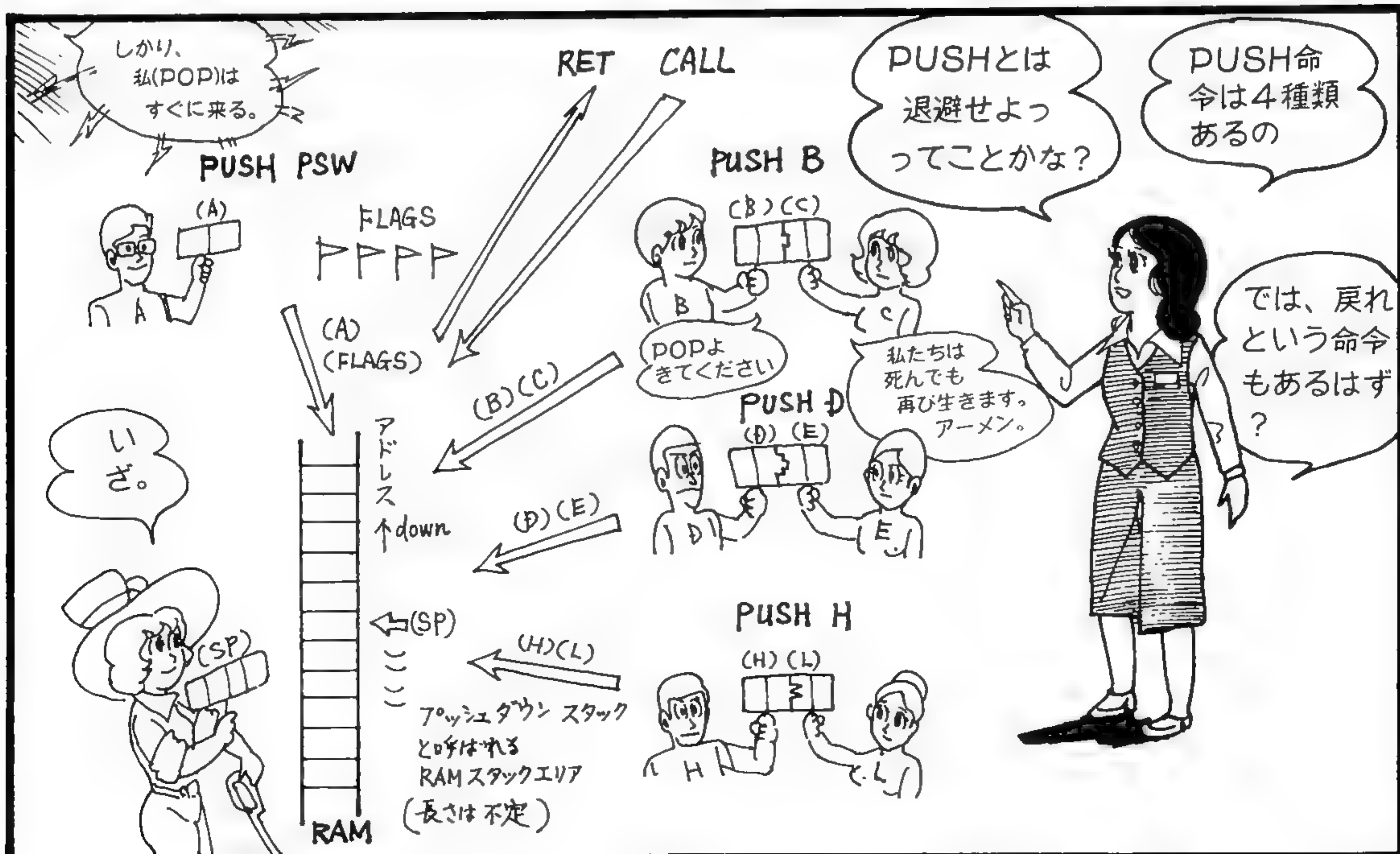
フッシュダウンスタックとは PUSH・CALL 命令でスタックポインタの示しているRAM領域のことです。PUSH命令を実行するとSPの内容は2つだけダウンするんです。

プログラム

LXI SP 20CD  
PUSH B  
PUSH D  
PUSH H  
PUSH PSW

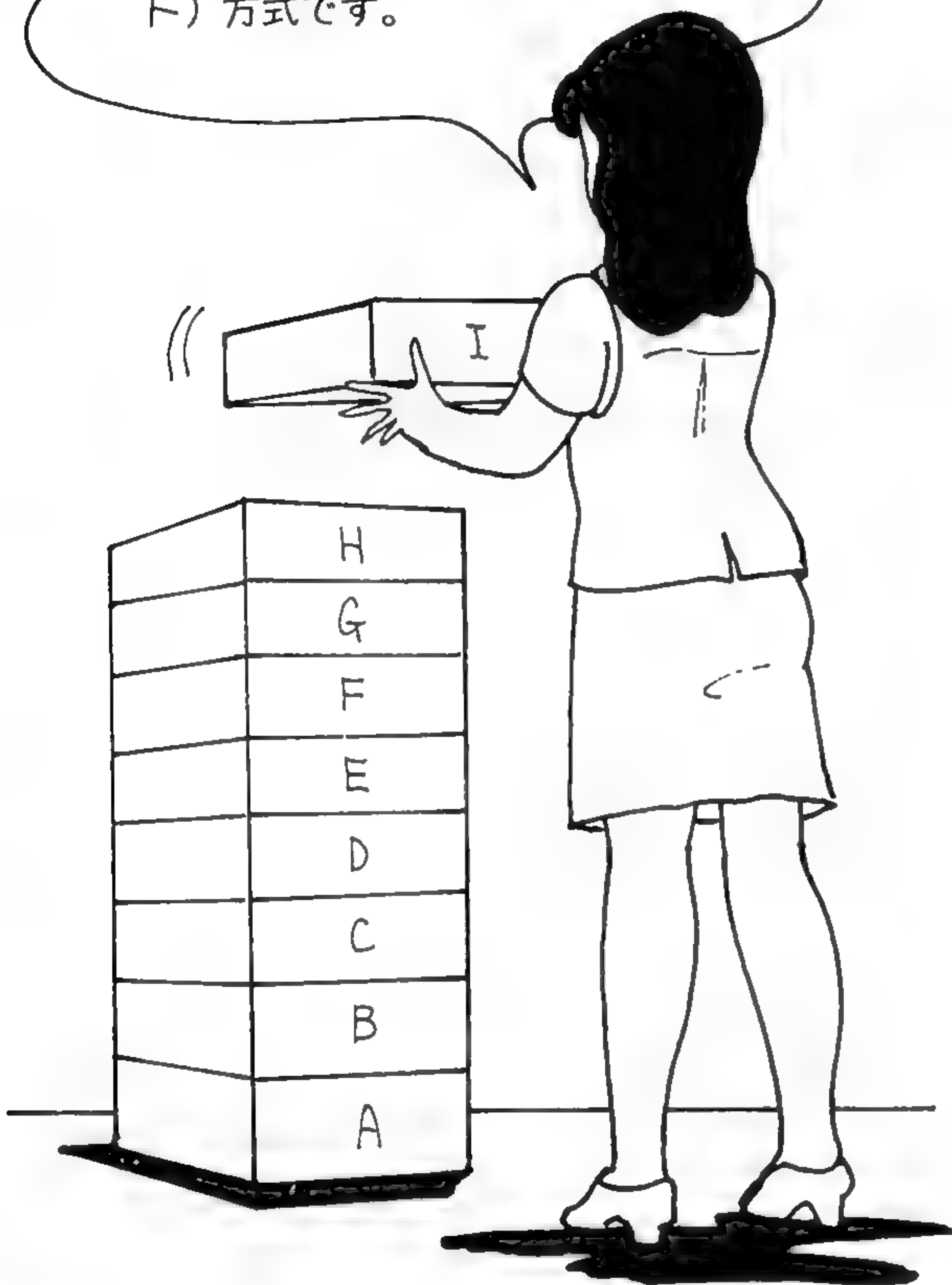


RAMメモリ



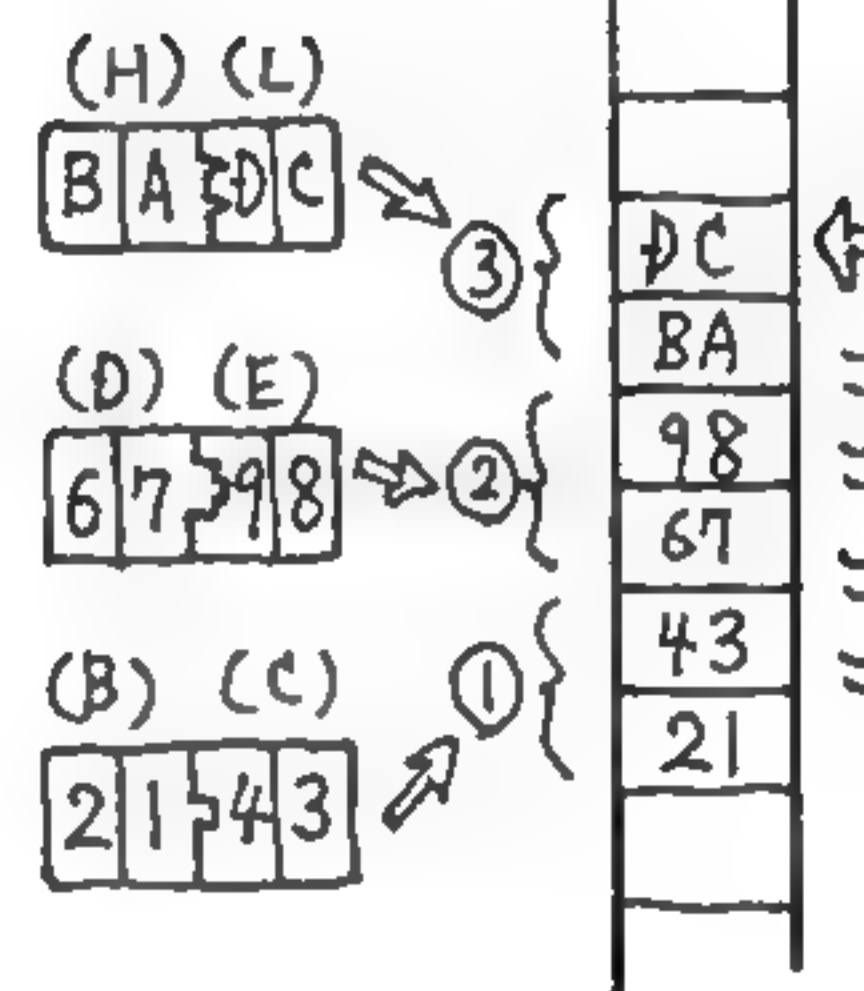


こういう積み方だと後に置いたものから先に取り出さねばならないですね。これがLIFO (ラストインファーストアウト) 方式です。



PUSH命令で退避させたデータを元に戻すのがPOP命令です。後から入れたものを先に出す(LAST IN FIRST OUT)しくみになっています。

プログラム  
① PUSH B  
② PUSH D  
③ PUSH H

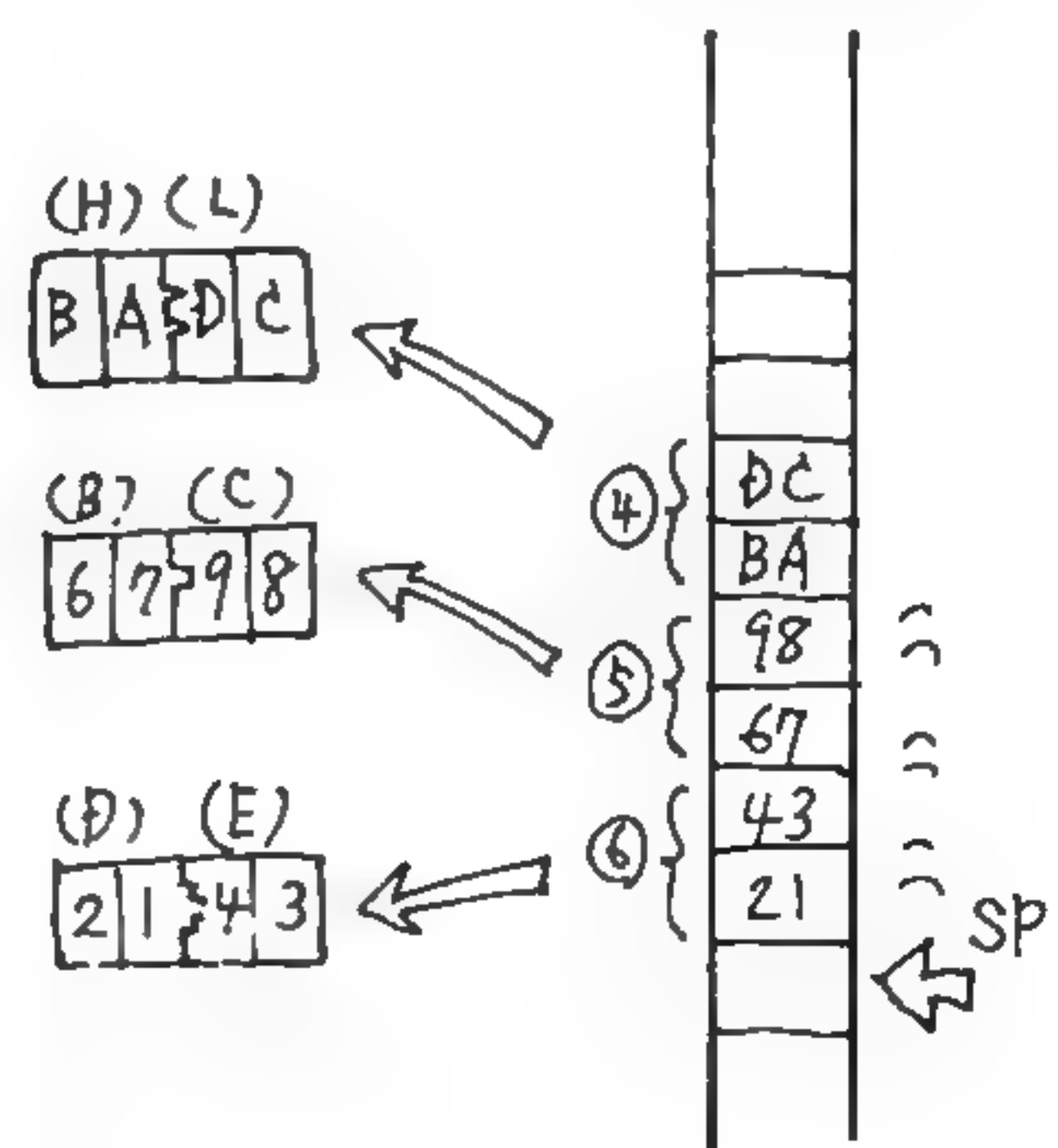


この場合は、HLレジスタペアからデータを戻します。順番を間違えると、データも別のものが入ることになります。

プログラム  
④ POP H  
⑤ POP B  
⑥ POP D

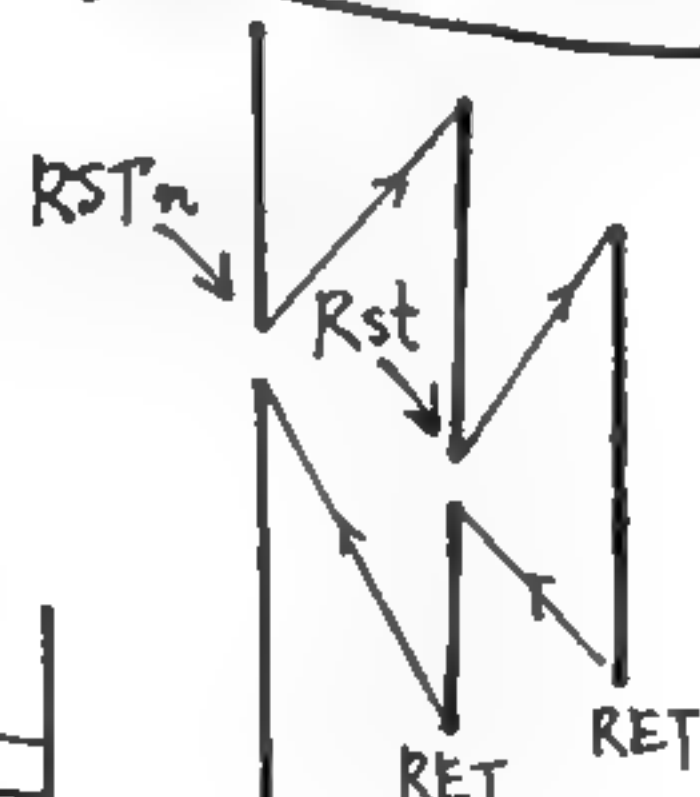
PUSHとPOPは対して使われるのです。

順番を間違えたか、またはわざとデータチェンジのために入れ換えたのかどちらか。



この場合も、フッシュダウンスタックに、戻る番地を入れておき、割り込みプログラムの終わりに書かれたRET命令によってメインに帰ります。

RST命令は、ハード的に割り込みポートから挿入される1バイトのベクトルアドレスジャンプ命令です。



ああ忙しい





## ⑧マシン語で プログラムだ!

$$\begin{array}{r} 342857 \\ + 183918 \\ \hline \end{array}$$

人間がやる方がずっと  
簡単だけど...



続いて加算数3バイトを  
2003番地からメモります。

2000	57	
2001	28	
2002	34	18 39 18
2003	18	
2004	39	
2005	18	

プログラムの内容は被加算数  
3バイトを2000番地から  
メモリ、

アドレス	
2000	57
2001	28
2002	34
	34 28 57

1メモリは1  
バイトに対応

と言っても動くわけないわね。  
マイコンにわかるマシン語で  
のプログラムにチャレンジす  
るしかないようね。



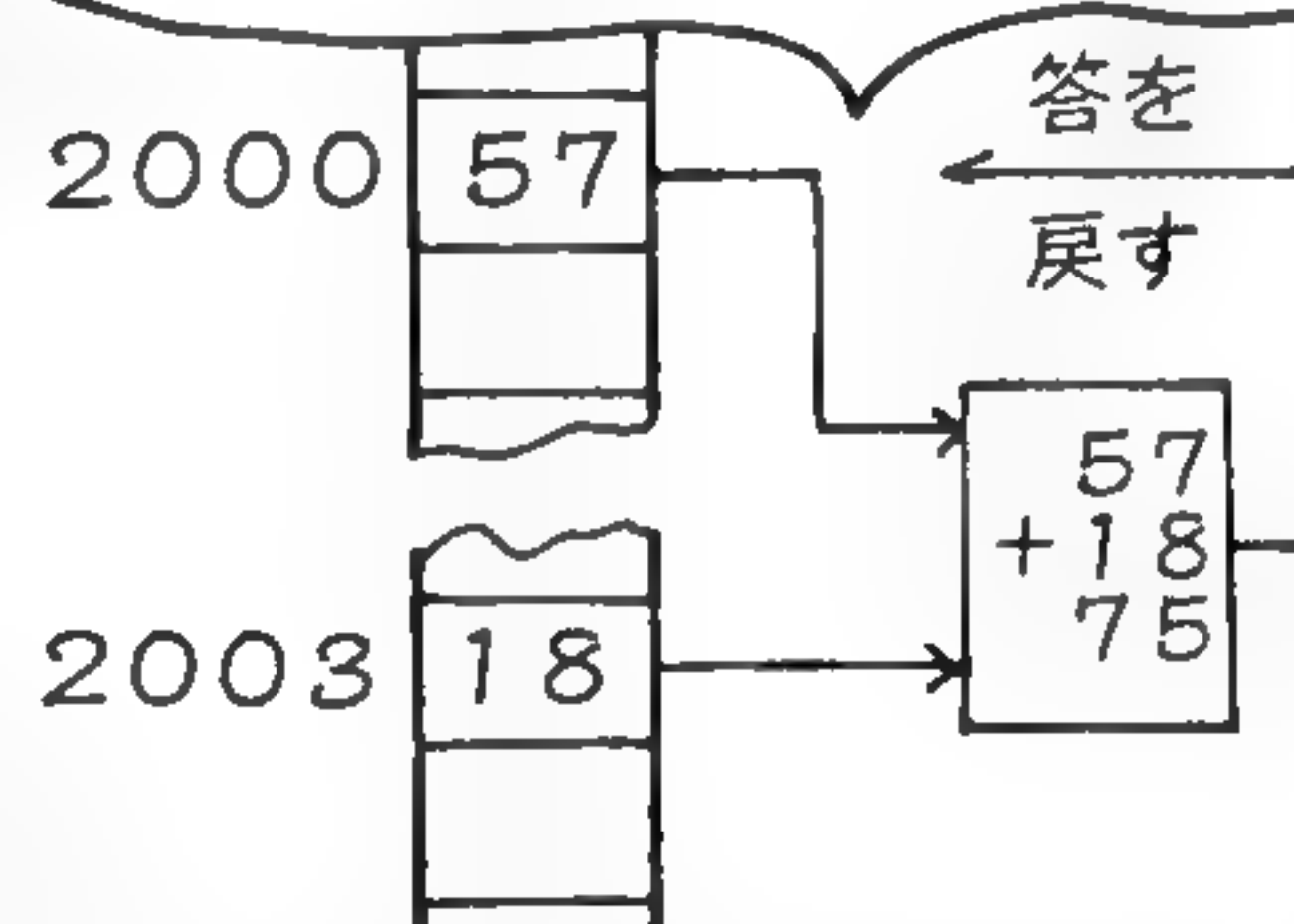
ここではLEDや画面に答を  
表示させるところまではやり  
ませんでした。OUTPUTサ  
ブルーチンを使えばできるは  
ずです。



答は被加算数の3バイトと置  
きかわって2000番地から  
の3バイトに入っていること  
になります。

2000	75	57
2001	67	28
2002	52	34
答		

次に下桁から1バイトずつた  
し算し、10進補正して結果を  
被加算数を入れておいたメモ  
リに戻します。



ハンドアセンブルの場合、コ  
ーディングの形式なんてどう  
でもいい気になりますが正式に  
やっておいた方があとでアセ  
ンブラを使う時、二度手間に  
ならず済みます。

スペース ↓ コメント  
↑  
LOOP: LDAX B;  
↑ ↑  
ラベル ニーモニック オペランド

人手でニーモニック → マシ  
ン語に変換することをハンド  
アセンブルといいます。

LXI B, 2000 → 210320  
LXI H, 2003 →

ニーモニック  
オペコード  
変換が必要

プログラムはまずニーモニッ  
クでコーディング(記述)し、  
それからマシン語に変換しま  
す。

ORG 2011  
LXI B, 2000  
LXI H, 2003  
XRA A  
MVI E, 03  
...



## プログラム例

## アセンブリ言語

ラベルはジャンプ先のアドレスを記号で表わすやり方でアセンブリ言語のコーディングがしやすくなります。

```

LOOP: LDAX B
      JUMP LOOP
  
```

マシン語のアドレスがわからなくてもコーディングできる。

では1行ずつハンドアセンブルしていきましょう。2011番地に最初のオペコード01が入ります。



ラベル	ニーモニック	オペランド	コメント
	ORG	2011	;最初のオペコード
①	LXI	B, 2000	;被加算数のアドレス
②	LXI	H, 2003	;加算数のアドレス
③	XRA	A	;キャリーのクリア
④	MVI	E, 03	;ループ回数のセット
⑤	LOOP: LDAX	B	;Aレジスタにロード
⑥	ADC	M	;加算
⑦	DAA		;10進補正
⑧	STAX	B	;Aレジスタの答を移す
⑨	DCR	E	; (E) = (E) - 1
⑩	JZ	DONE	; (E) = 0でDONEへ
⑪	INX	B	; (BC) = (BC) + 1
⑫	INX	H	; (HL) = (HL) + 1
⑬	JMP	LOOP	;上桁計算へ
⑭	DONE: HLT		

このプログラムを走らせる前にデータをキーインしておかなくてはなりません。答もキー操作でアドレスのデータで確認します。

アドレス データ

2000-57 2003-18  
2001-28 2004-39  
2002-34 2005-18

加算数の下位番地はHLレジスタに入れます。

```

LXI H, 20 03
21 03 20
  
```

マシン語です。

番地 データ  
2014 210320

「」はスペースを示します。

```

LXI B, 20 00
01 00 20
  
```

まず被加算数の下位番地をBCレジスタに入れます。これは3バイト命令です。

番地 データ  
2011 010020

20 00  
(B) (C)

```

MVI E, 03
1E 03
  
```

計算回数をセットします。

④  
番地 データ  
2018 1E 03

3回目 2回目 1回目  
34 28 57  
+ 18 + 39 + 18

これは1バイト命令です。

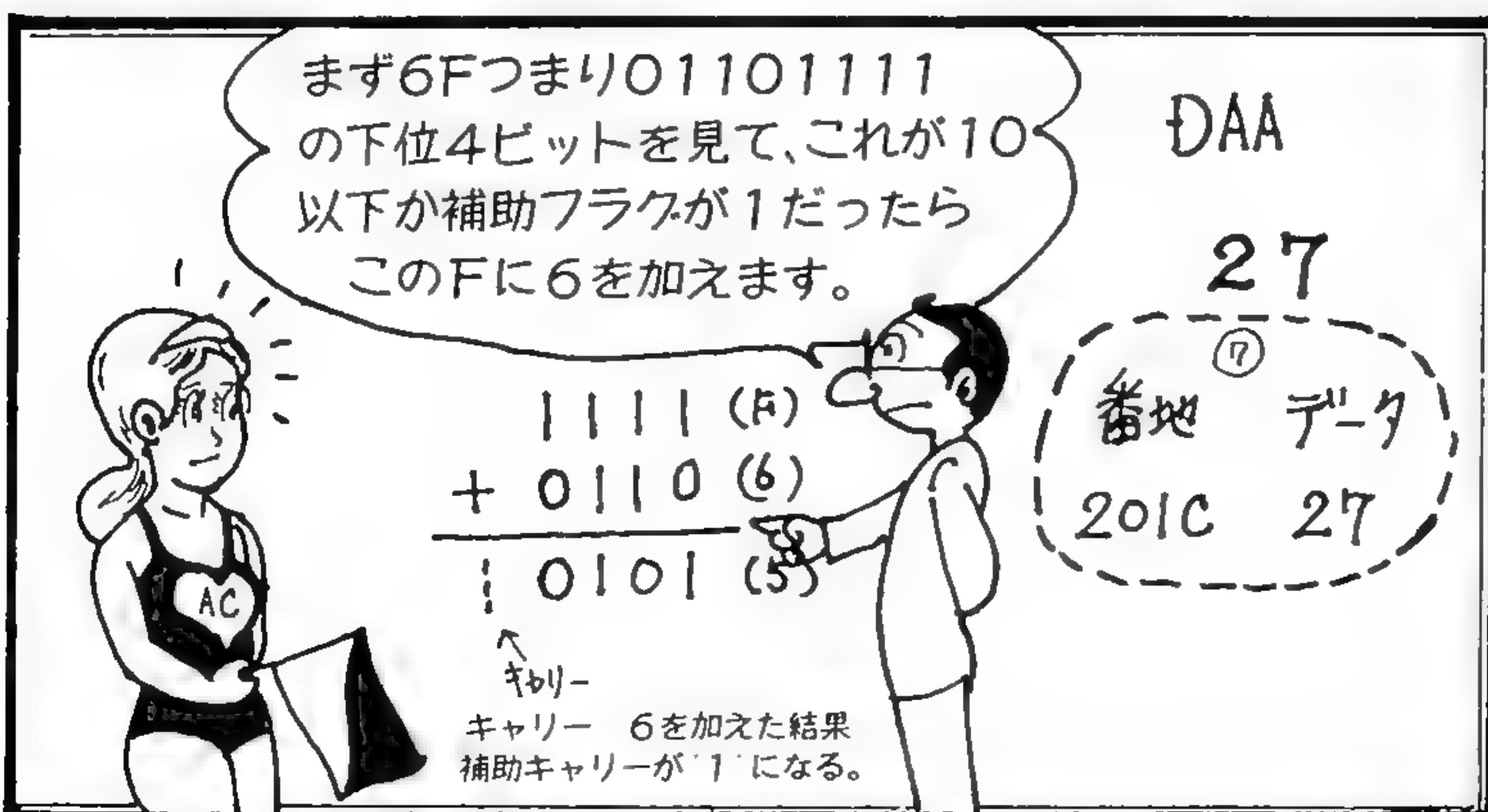
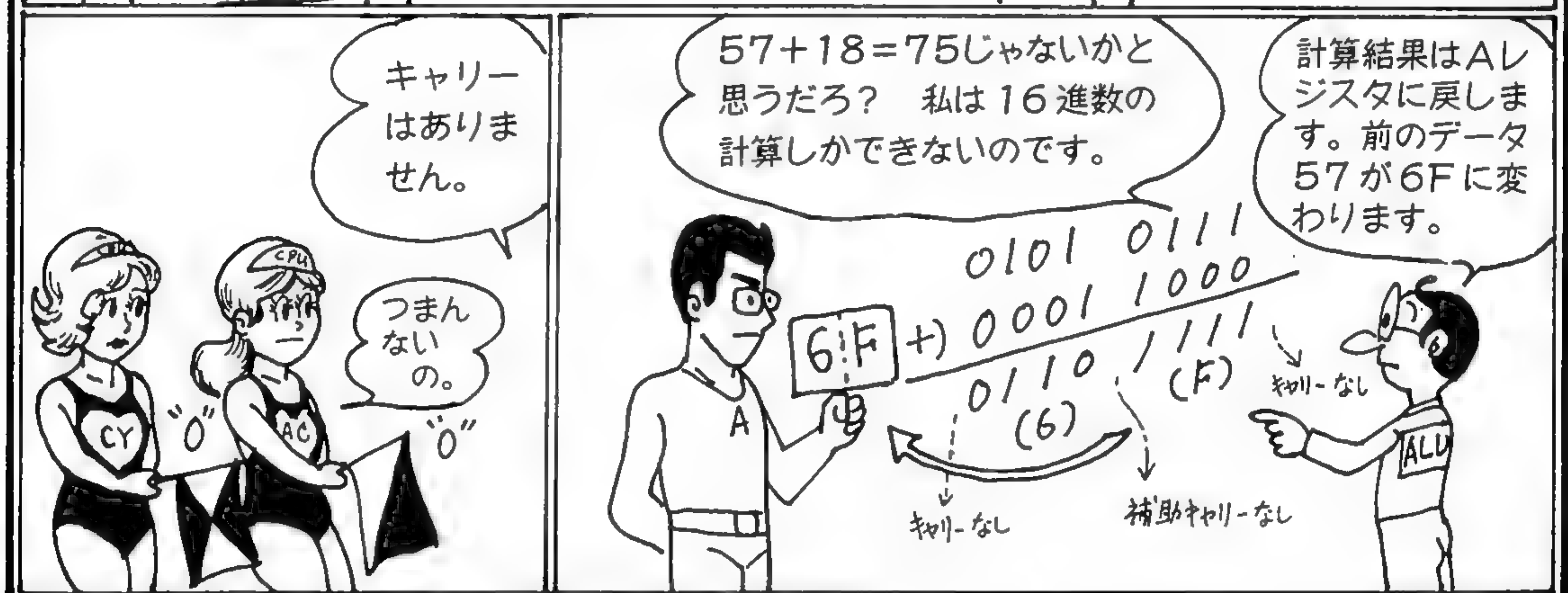
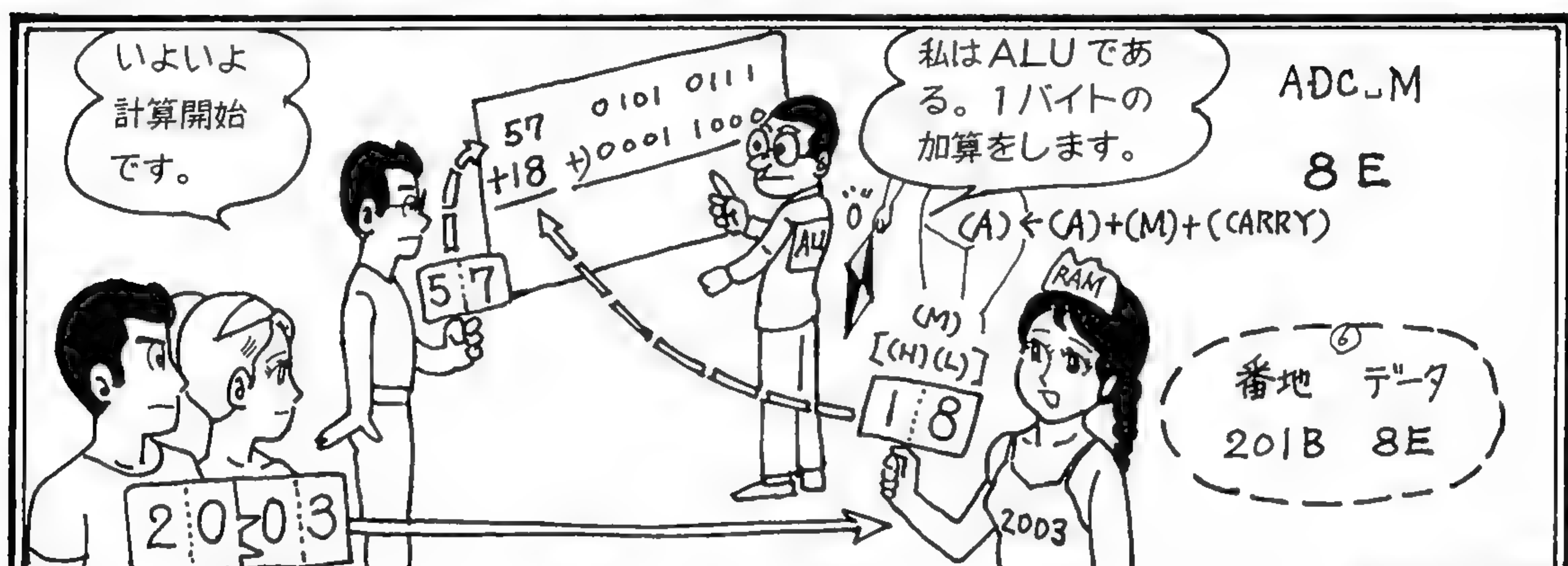
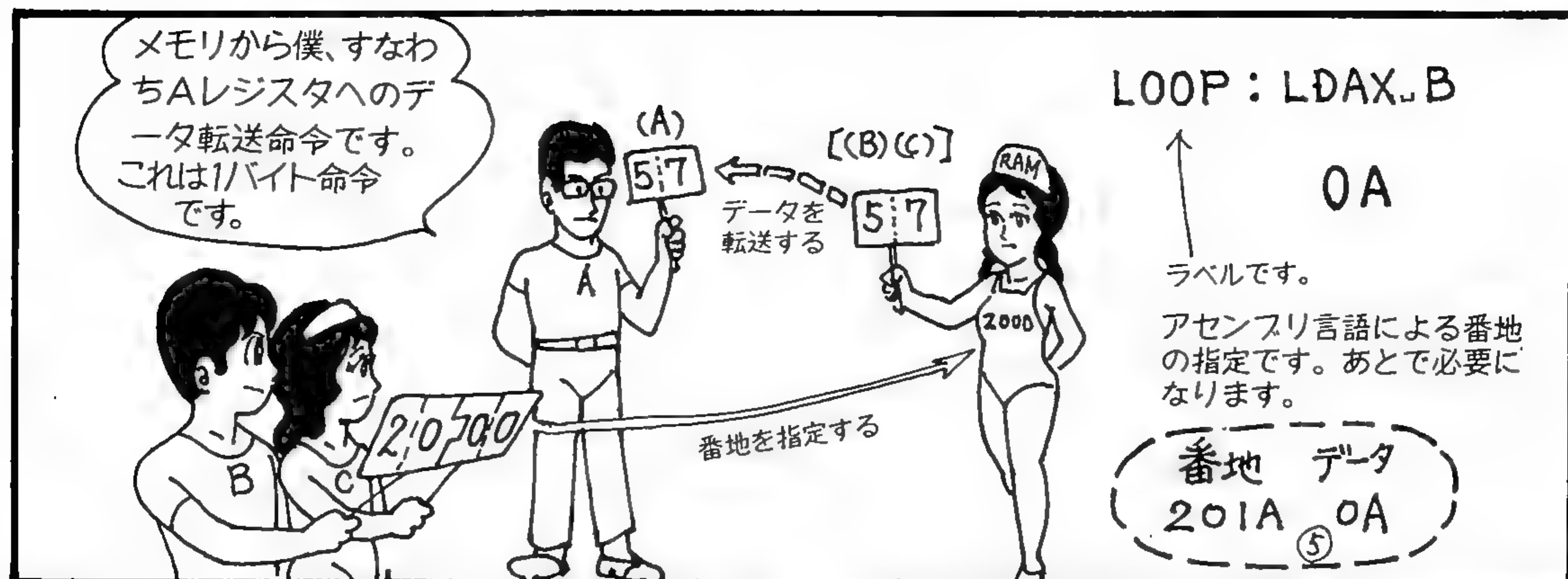
キャリーの補助フラグと桁上げフラグをリセットします。

```

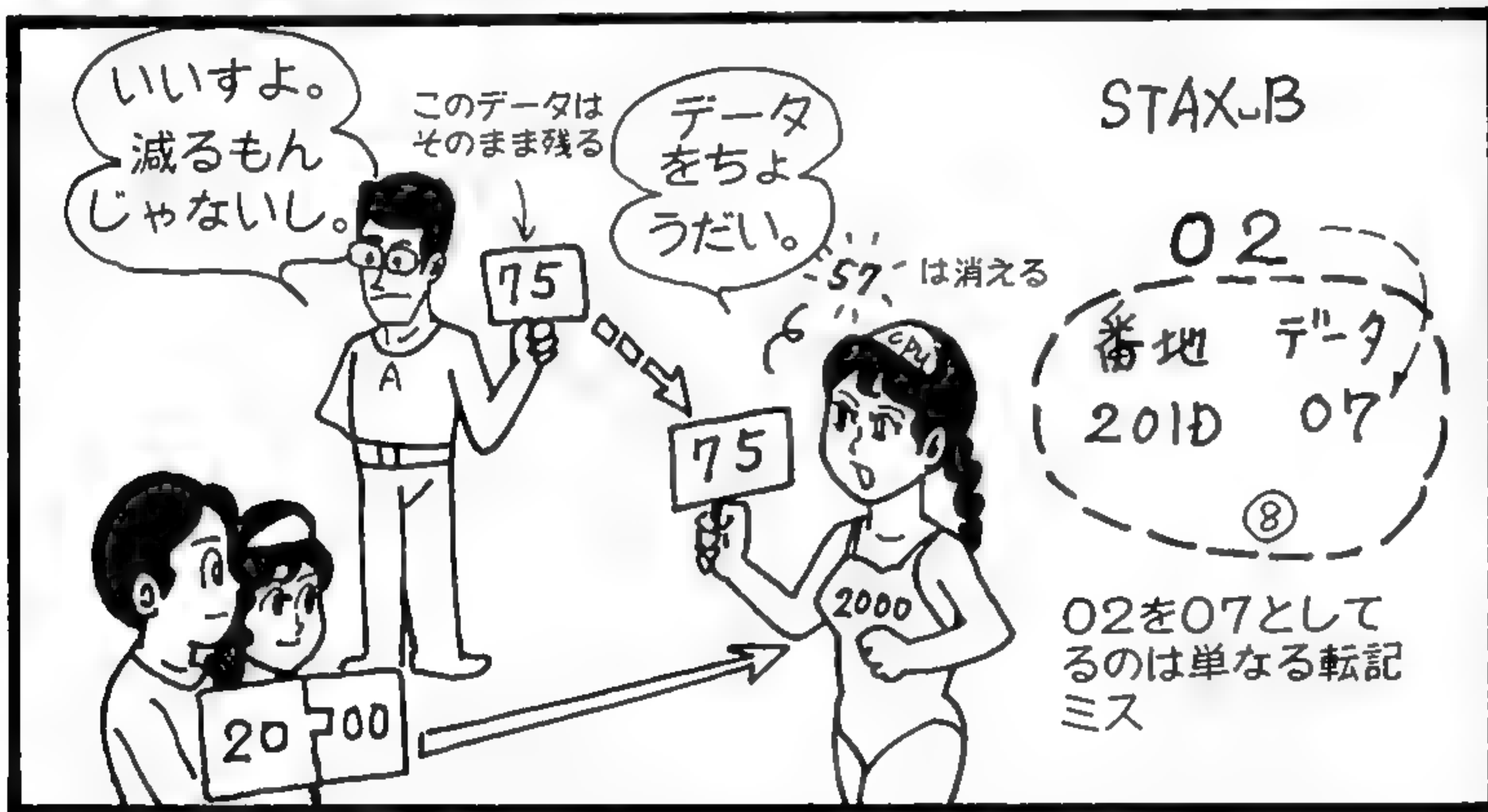
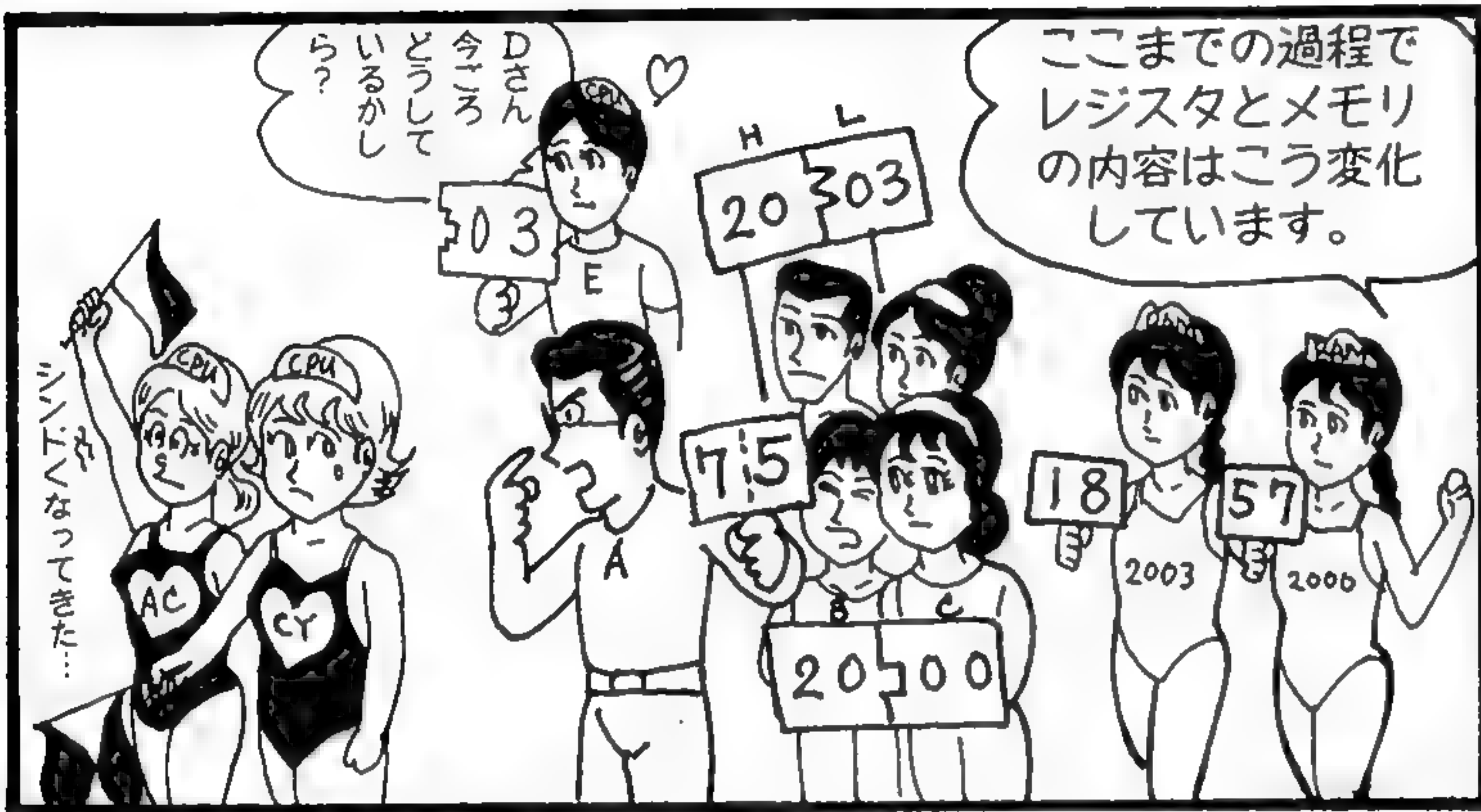
XRA A
  
```

A ← E  
番地 データ  
2017 AF

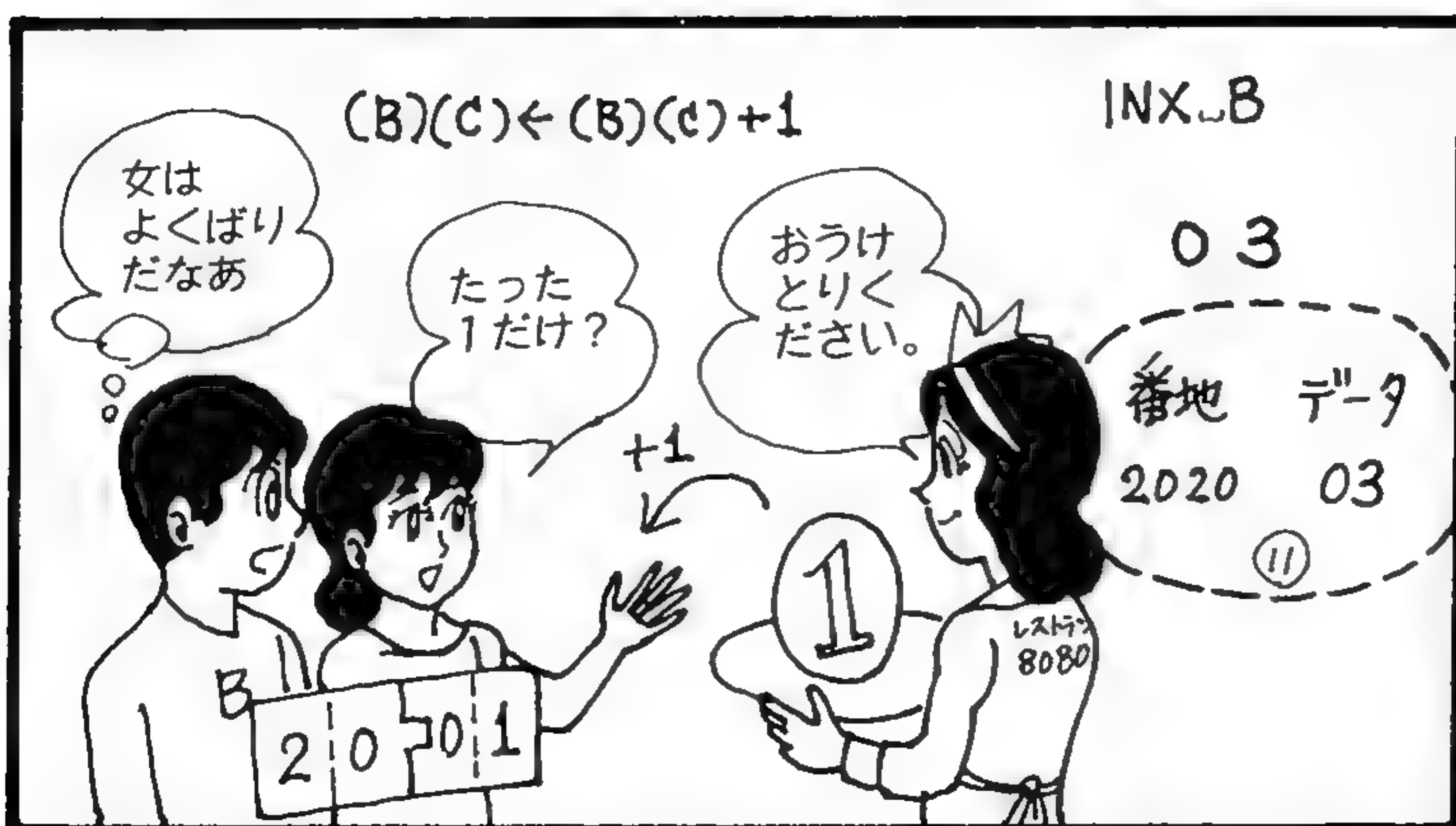
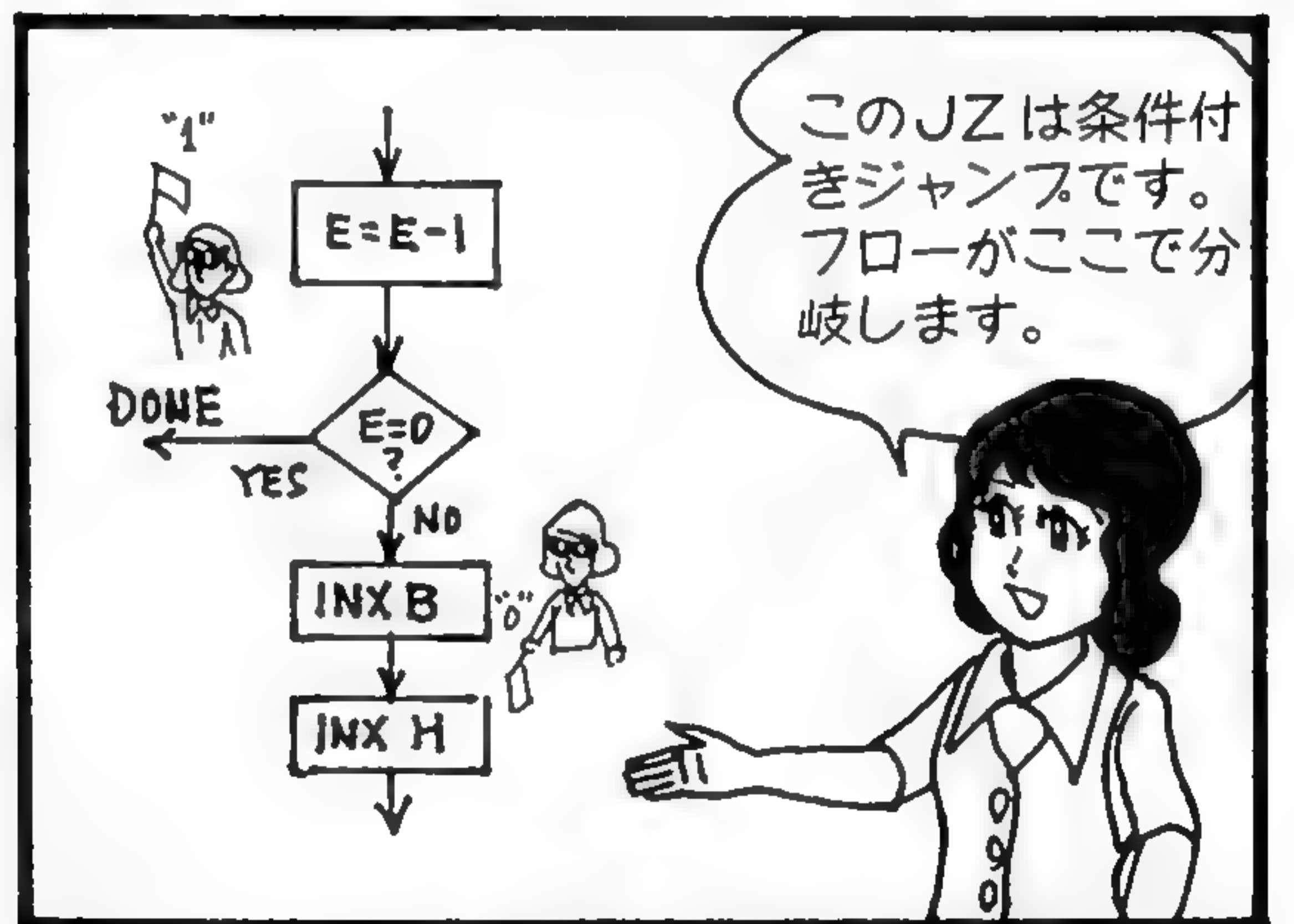
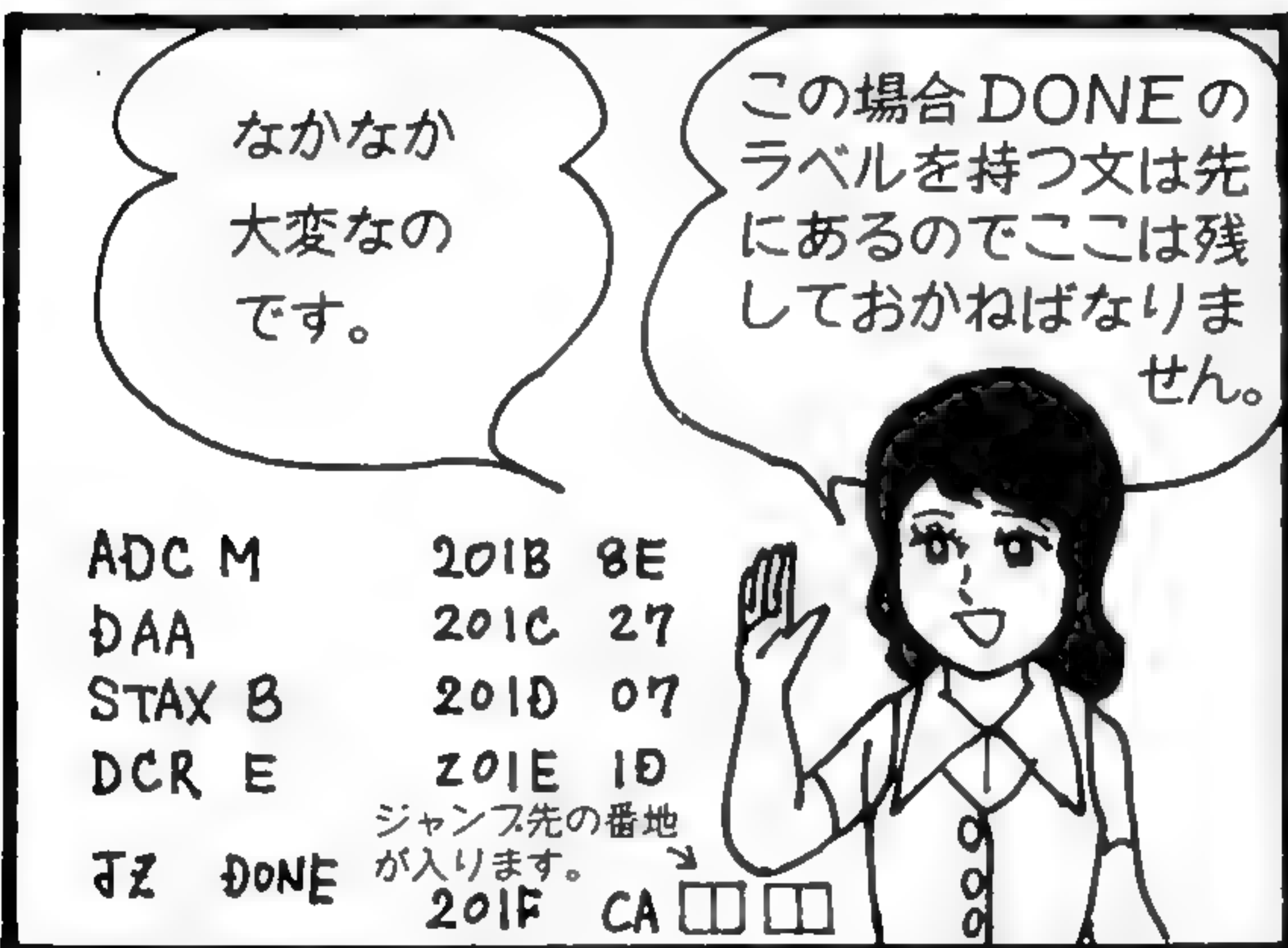














よろしく。私は無条件ジャンプ司令官です。このループも番地を示すラベルです。

JMP LOOP  
C3        
          B2 B3

Loop (B3) (B2)  
2 0 1 A

番地 データ (13)  
2024 C3 1A 20

(H)(L) ← (H)(L) + 1      INX H

+1はインクリメントというんだね。

ごろうさま。

あ、これ。

番地 データ (12)  
2023 23

20 01  
(H) (L)

+1

マシン語に直すとこの201Aは1A20となります。

JMP 201A  
C3 1A 20

この理由はCPUが下位アドレスを先に読むからです。

C3
1A
20

↑ 下位アドレス

アセンブラ語      機械語

ラベル    ニーモニック	番地    データ
XRA A	2017 AF
MVI E 03	2018 1E 03
LOOP: LDAX B	201A 0A
ADC M	201B 8E
DAA	201C 27
...	
JMP LOOP	2024 C3 1A 20

このLOOPはすでにアセンブルした所にあります。この場合LOOPは201A番地です。

ここでさきほどのDONEの番地が決まりました。2027番地です。

うーん。何もしてはいけないんだって。

NOPではない

ねえどうしたの？

ラベル    ニーモニック  
DONE HLT  
          76

番地    データ (14)  
2027 76

これでハンドアセンブルが完了しました。

JZ DONE 201F CA 27 20

ハッ

きみ、これを

ごろうでした。

JZ 司令官に報告します。

DONE  
20 27



マシン語		翻訳	アセンブリ言語		
番地	データ		ラベル	ニーモニック	オペランド
		← ハンドアセンブル			ORG 2011
20 11	01 00 20				LXI B, 2000
20 14	21 03 20				LXI H, 2003
20 17	AF				XRA A
20 18	1E 03				MVI E, 03
20 1A	0A		Loop:		LDAX B
20 1B	8E				ADC M
20 1C	27	ホントは間違っている			DAA
20 1D	07				STAX B
20 1E	1D				DCR E
20 1F	CA 27 20				JZ DONE
20 22	03				INX B
20 23	23				INX H
20 24	C3 1A 20				JMP Loop
20 27	76				DONE: HLT
20 00	57 28 34				DB 57 28 34
20 03	18 39 18				DB 18 39 18

このマシン語プログラムはモニタ付きのパソコンでもアドレスを変えれば走ります。ただしCPUが8080かZ80の場合です。

PC-8001ならS、G、Dのコマンドを使います。

あなたも8080、Z80マシンでやってみてください。

このプログラムをパソコンでアセンブルするのはあとです。オールRAM型パソコンで、CP/Mプログラムの走るものを使います。ここではSDK-85を使いアドレスとデータを1つずつ示していきます。



わかりと思いますがG(ゴー)させる前にはC000番地から57、28、34、18、39、18のデータを入れておかないけません。



PC-8001ならORG C011としてもう一度ハンドアセンブルしてください。FM7/8などはアドレスだけでなくマシン語も変える必要があります。



57が入ります。

次に57と押すと2000番地に

2000 57

↑  
2000番地に57を入れようとしている。

2000番地が表示されます。

アドレスを表示する

データを表示する

2000 E5

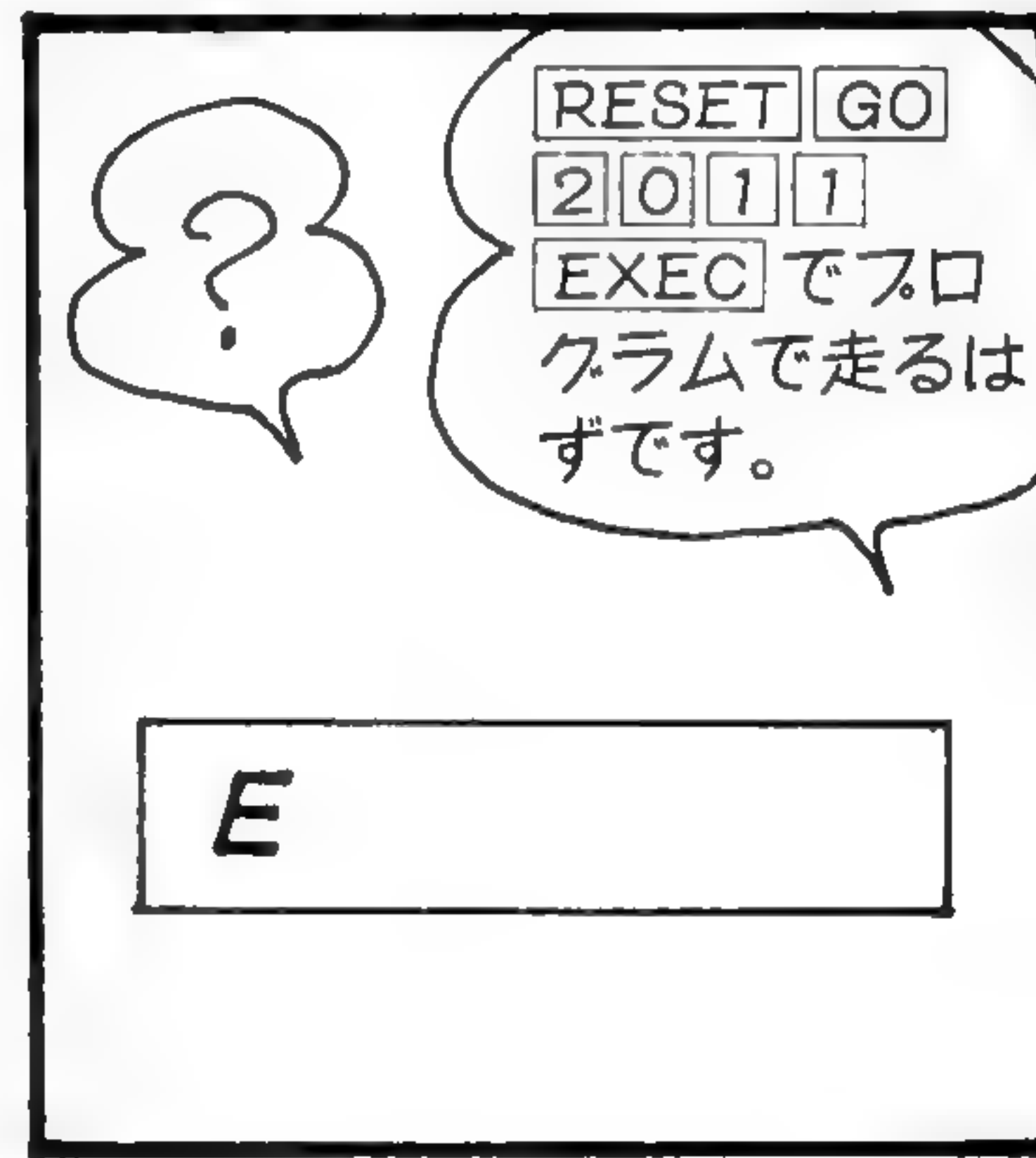
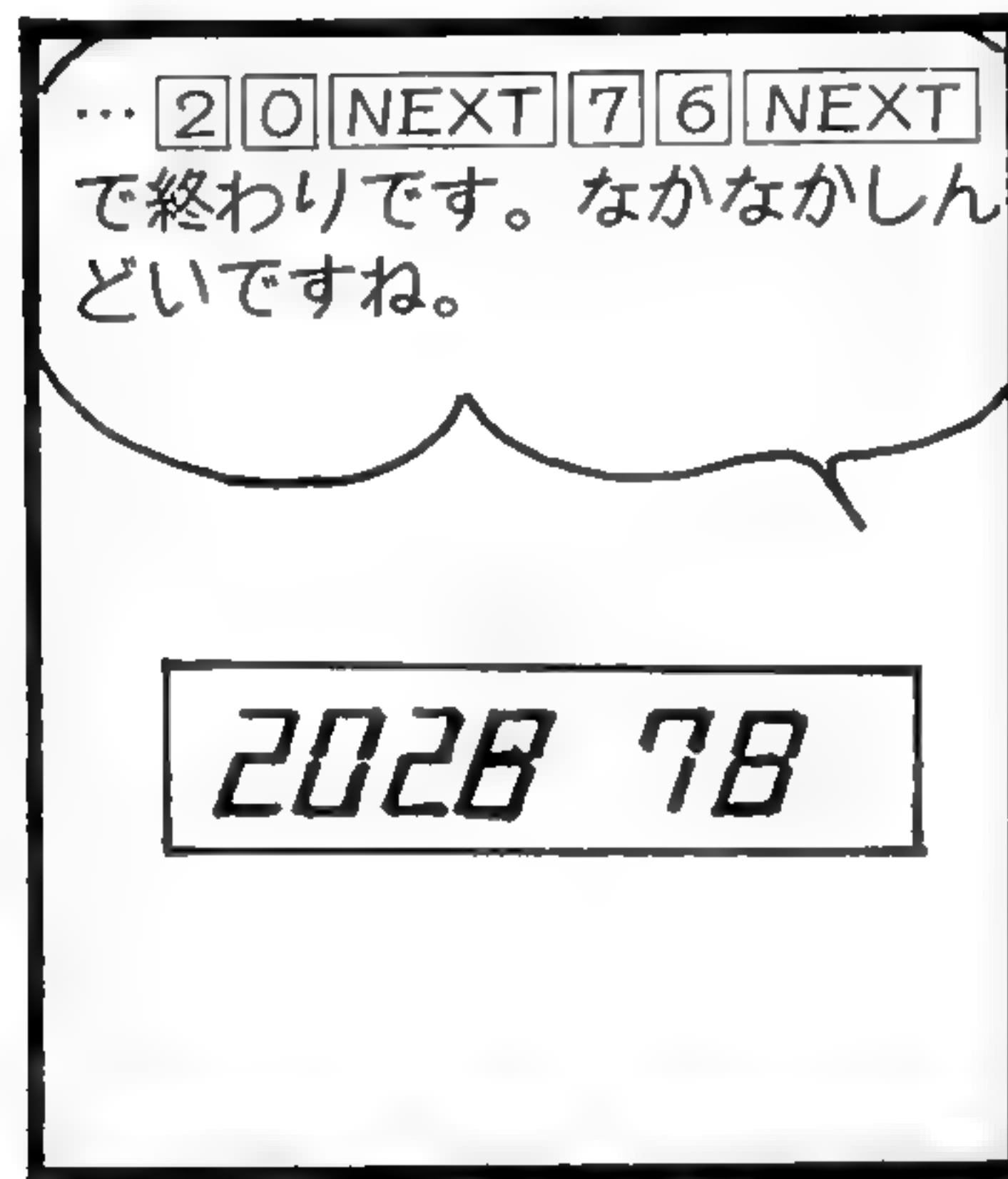
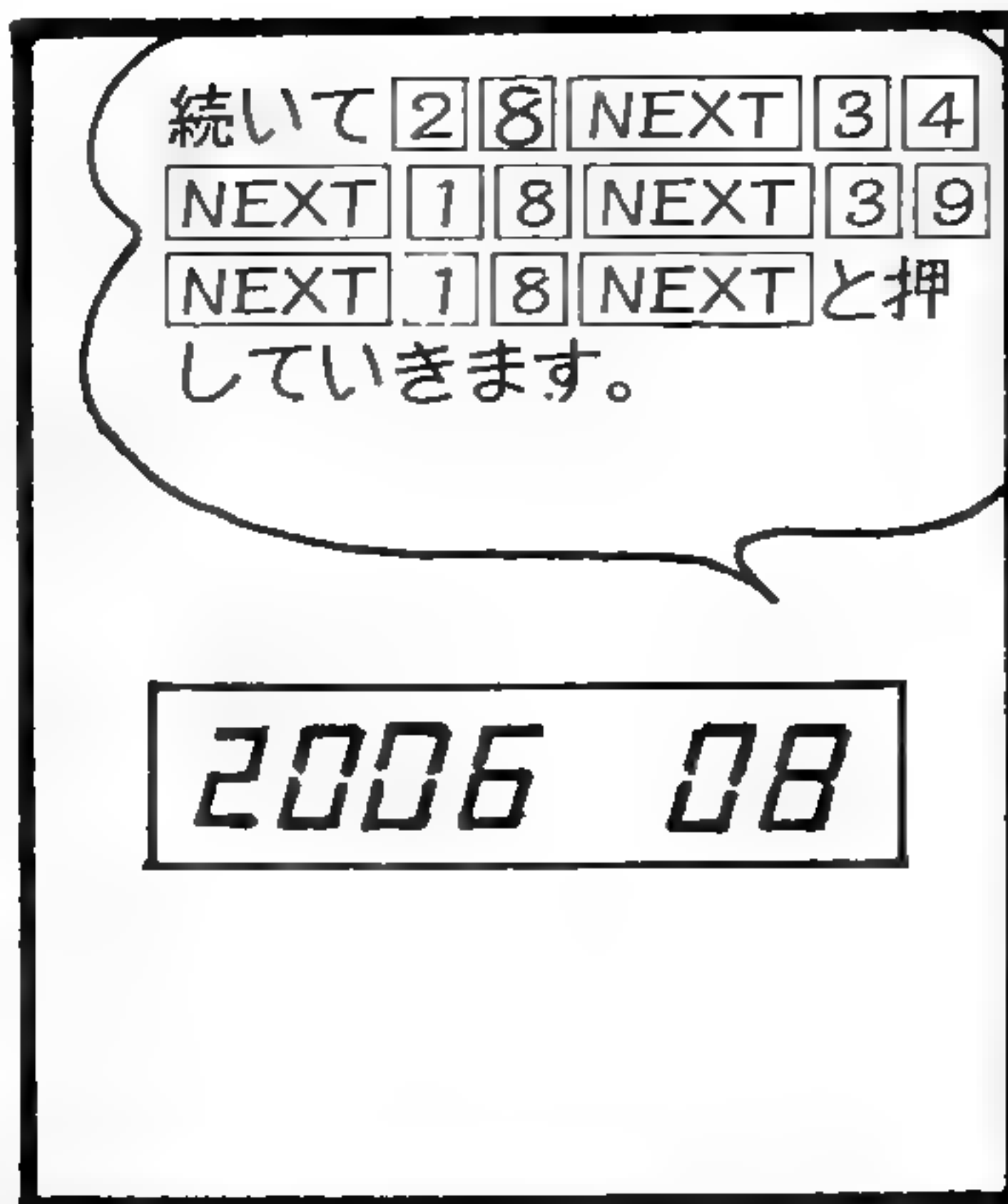
↑  
この表示は、前のものでキーインして57にします。

RESET SUBST MEM  
2 0 0 0  
NEXT とキーインしました。

まず、









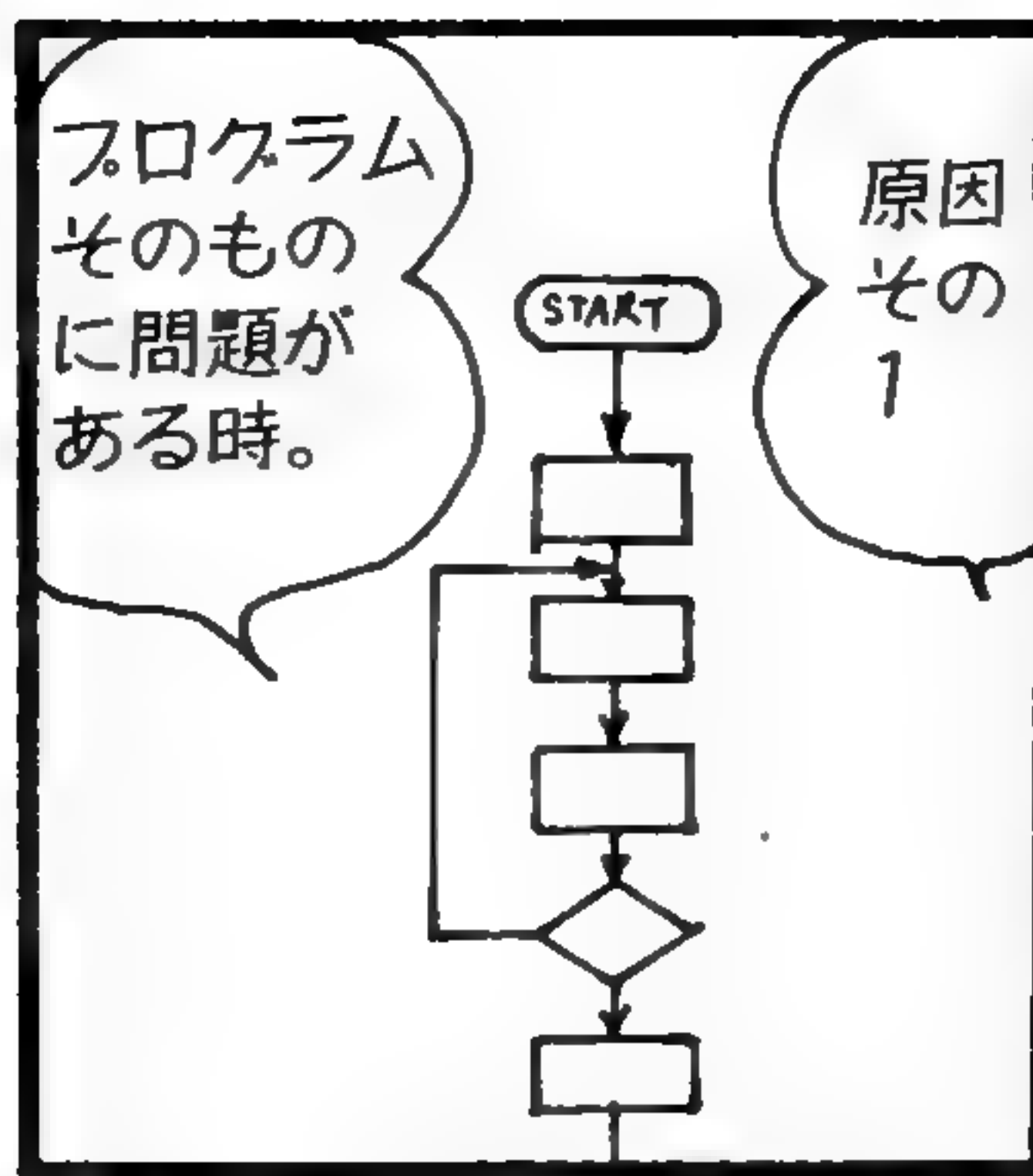
ハンドアセンブルの際の変換ミス。

原因  
その  
2

STAX B	番地	データ
	201D	07

↓ 02      ↑

こんな信じられないようなミスをすることもあります。



期待した通りにプログラムが走らなかった時、その原因を調べて誤りがあれば修正しなければなりません。

デバッグ作業  
(虫とり)

プログラム全体を通してこんなミスがあるものです。これを虫といい、ひとつづつつぶしていかなければなりません。

程度  
の差  
こそ  
あれ  
...

もう一度リストを見ると、ありましたね。  
印の07は実は02でした。

キーインする時のミス、などがあると思います。

原因  
その  
3

するとただちにEとなる。

さてもう一度アタック。RESET GO 2 0 1 1 EXEC と押す...

E

RESET SUBST MEM 2 0 1 D NEXT 0 2 NEXT で修正完了。

この表示は、このキーを押した時のものです。

201D 07

↑  
これはDです。

その他のミスはなさそうですから201D番地の内容を変更します。

52! やった!

NEXT

$$\begin{array}{r} 342857 \\ + 183918 \\ \hline 526775 \end{array}$$

2002 52

67!

NEXT

$$\begin{array}{r} 342857 \\ + 183918 \\ \hline 6775 \end{array}$$

2001 67

+1ずつ増えていくのをインクリメントといいます。

いいぞ。

RESET SUBST MEM 2 0 0 0 NEXT

むむ、75だ!

$$\begin{array}{r} 342857 \\ + 183918 \\ \hline 75 \end{array}$$

2000 75



状態数を計算すればおよその時間がわかるのです。今SDK-85の1状態分の時間がわかりませんのでNECのTK-80の時間でやってみます。

キット TK-80の  
1状態は、  
0.4882812  $\mu$  SEC

この計算は何秒かかったのでしょうか。

こんどはバッチリでした。

ところで。

ニーモニック	ステート	回数	計
LXI B	10	1	10
LXI H	10	1	10
XRA	4	1	4
MVI E	7	1	7
LDAX B	7	3	21
ADC M	7	3	21
DAA	4	3	12
STAX B	7	3	21
DCR E	5	3	15
INX B	10	4	40
INX H	5	3	15
JMP	10	3	30
HLT	7	1	7

ここでの回数はループのルーチンで何回も同じ命令が実行されたことを示すものです。状態数の合計は各命令の実行回数を考えねばなりません。

フワア  
ねむいニヤ

状態数は命令によって決まっています。

プログラムのメモリの容量と、実行時間とは関係ないのでご注意ください。

合計 288ステート  $\times$  0.4882812 = 140.62498  $\mu$  SEC

そうじ、せんたく、料理ができるかよ。

チッおまえに

0.00014 SEC

342857  
+ 183918

おれなら5秒はかかるナ...

(この時分作者はチョンガーでした)

このスピードと信頼性がとても魅力的です。

ピンとこないけど...

このプログラムは288ステート必要で実行時間は約140  $\mu$  SECです。すごいスピードですね。

1  $\mu$  SEC =  $\frac{1}{1000000}$  SEC

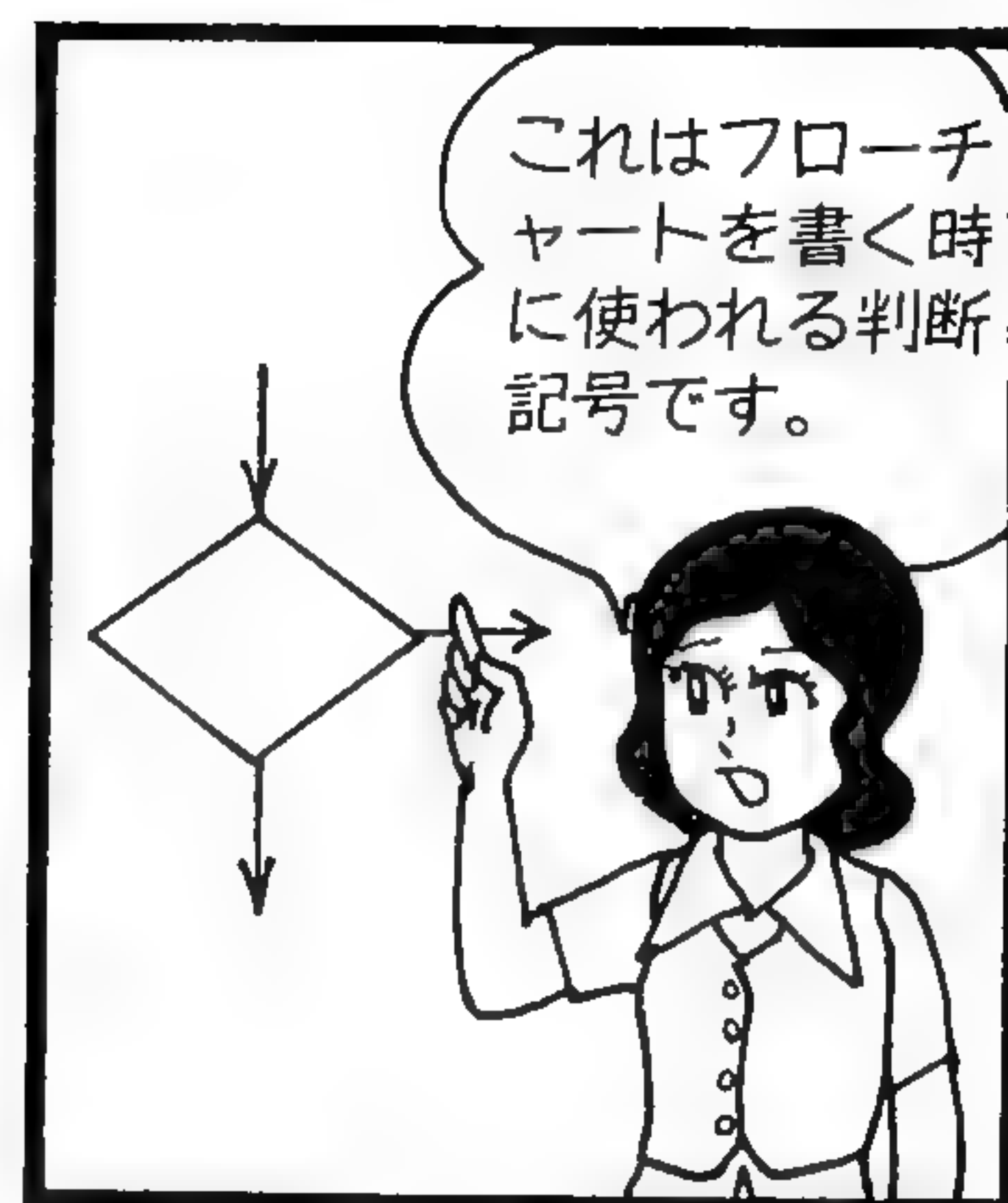
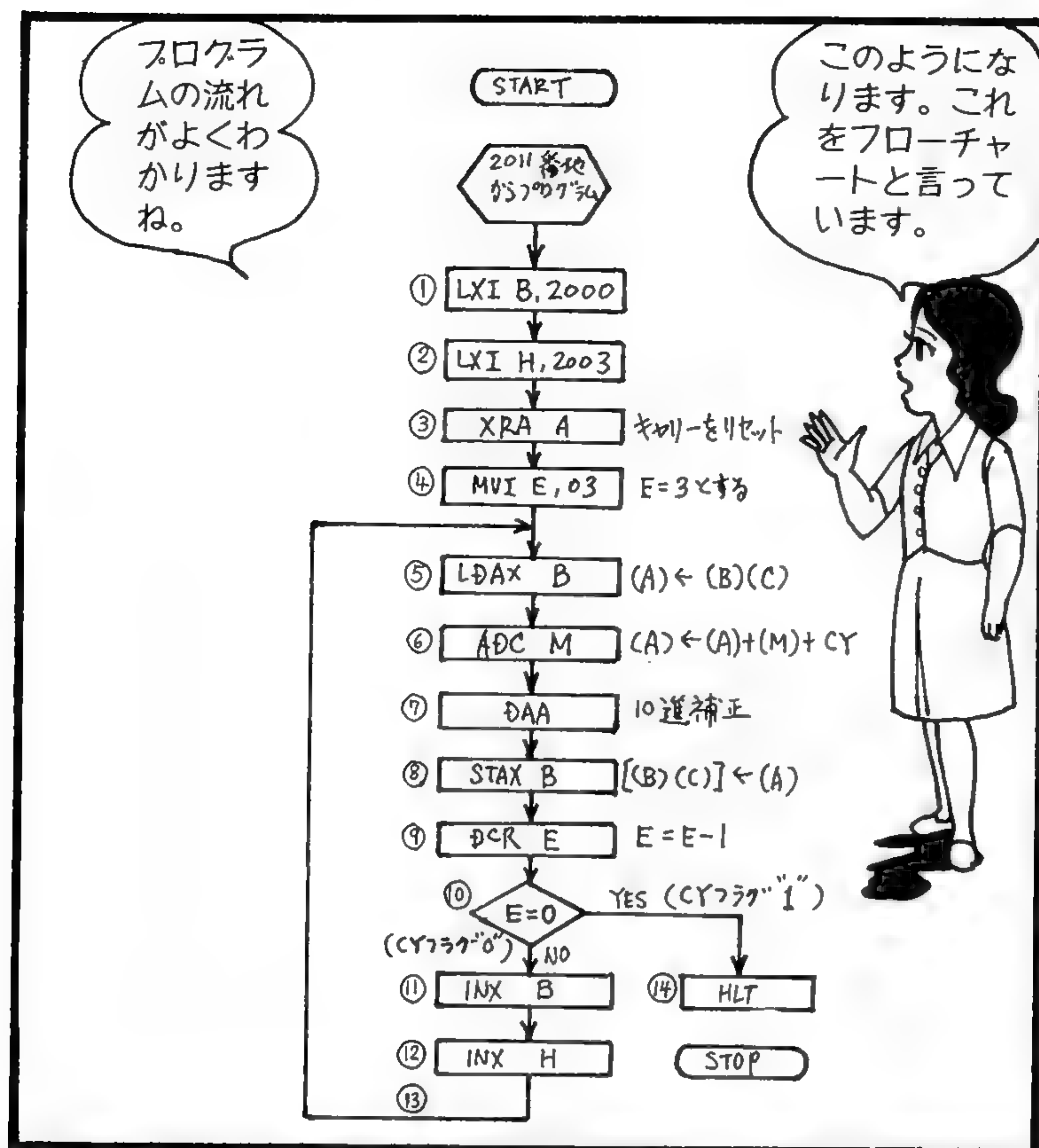
マイクロ秒

判断機能は、フラグ操作命令とその次に置かれる条件付きジャンプ命令との組合せでできます。

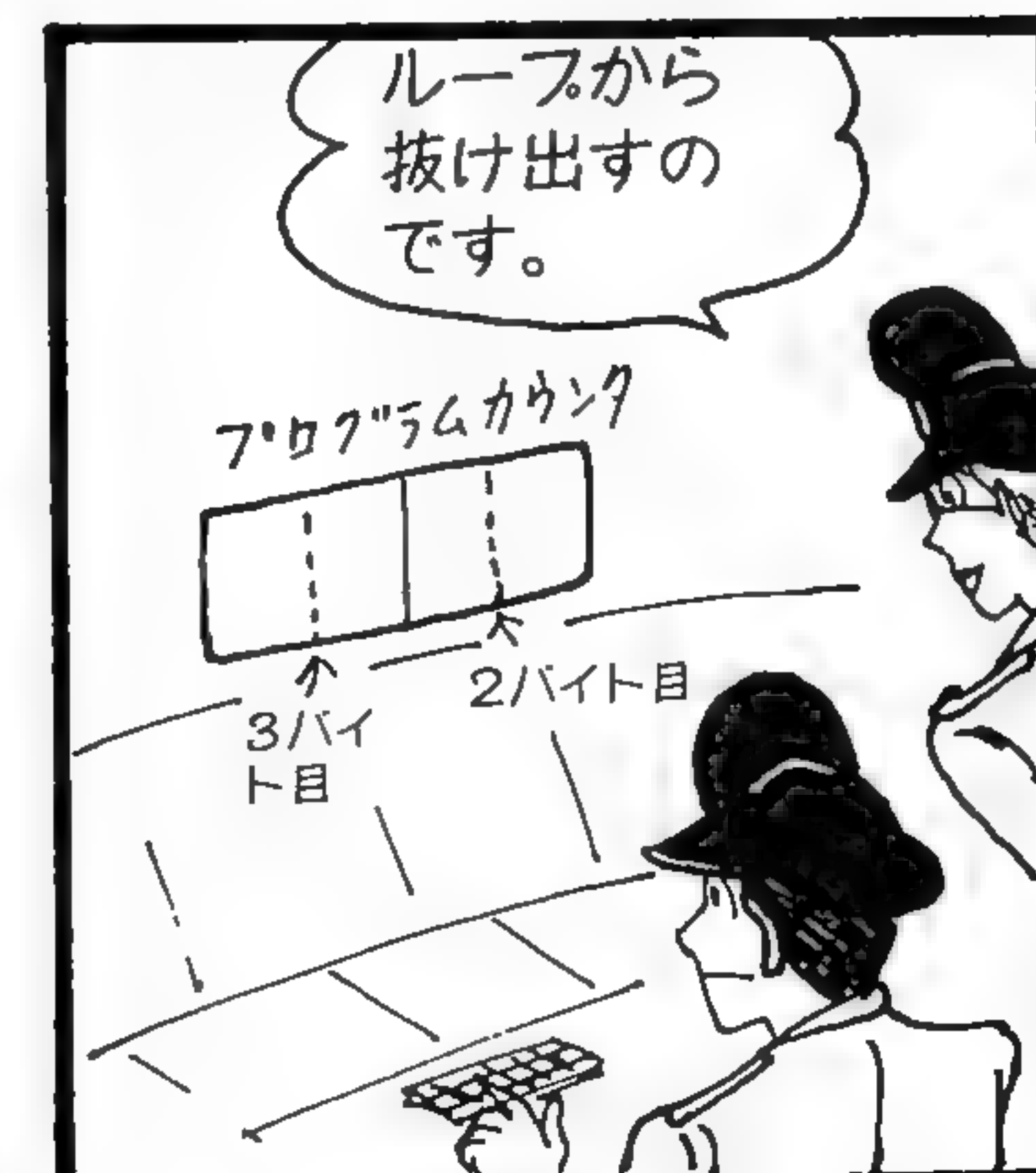
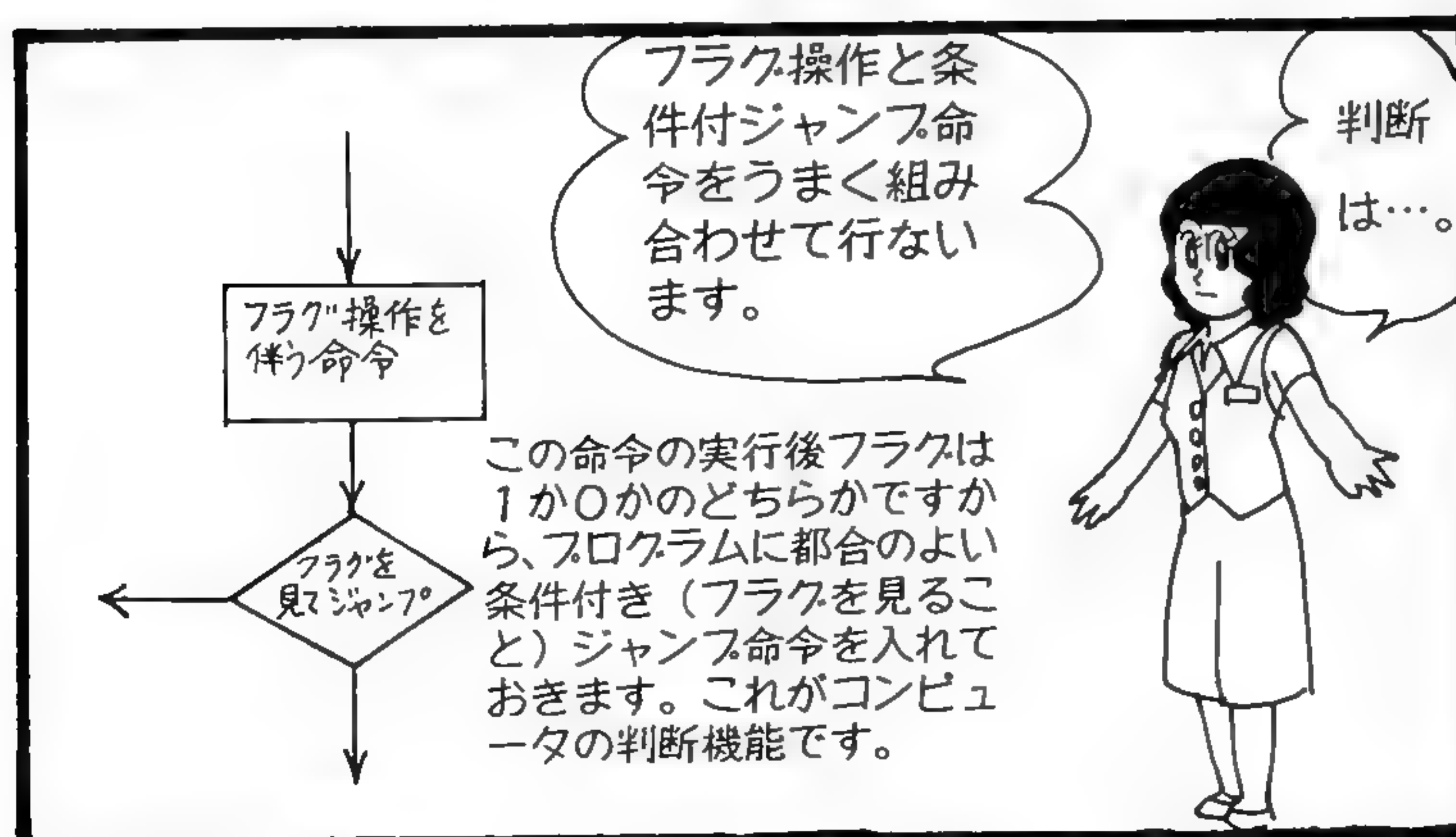
この判断機能があるため昔は電子頭脳なんて言われたわけです。そのメカニズムをもう少し説明しましょう。

それにもう1つの大きな魅力は判断機能があることです。







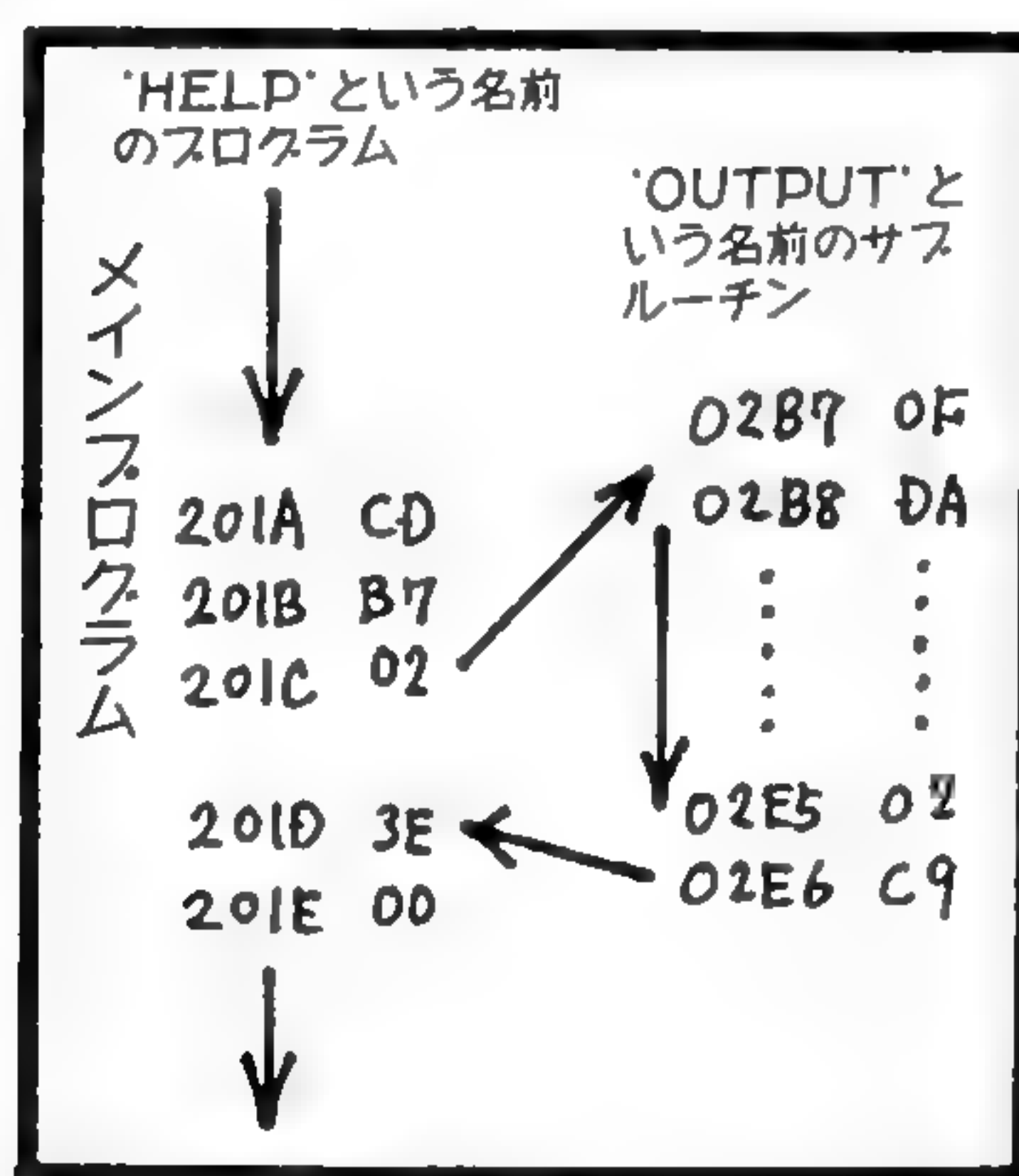
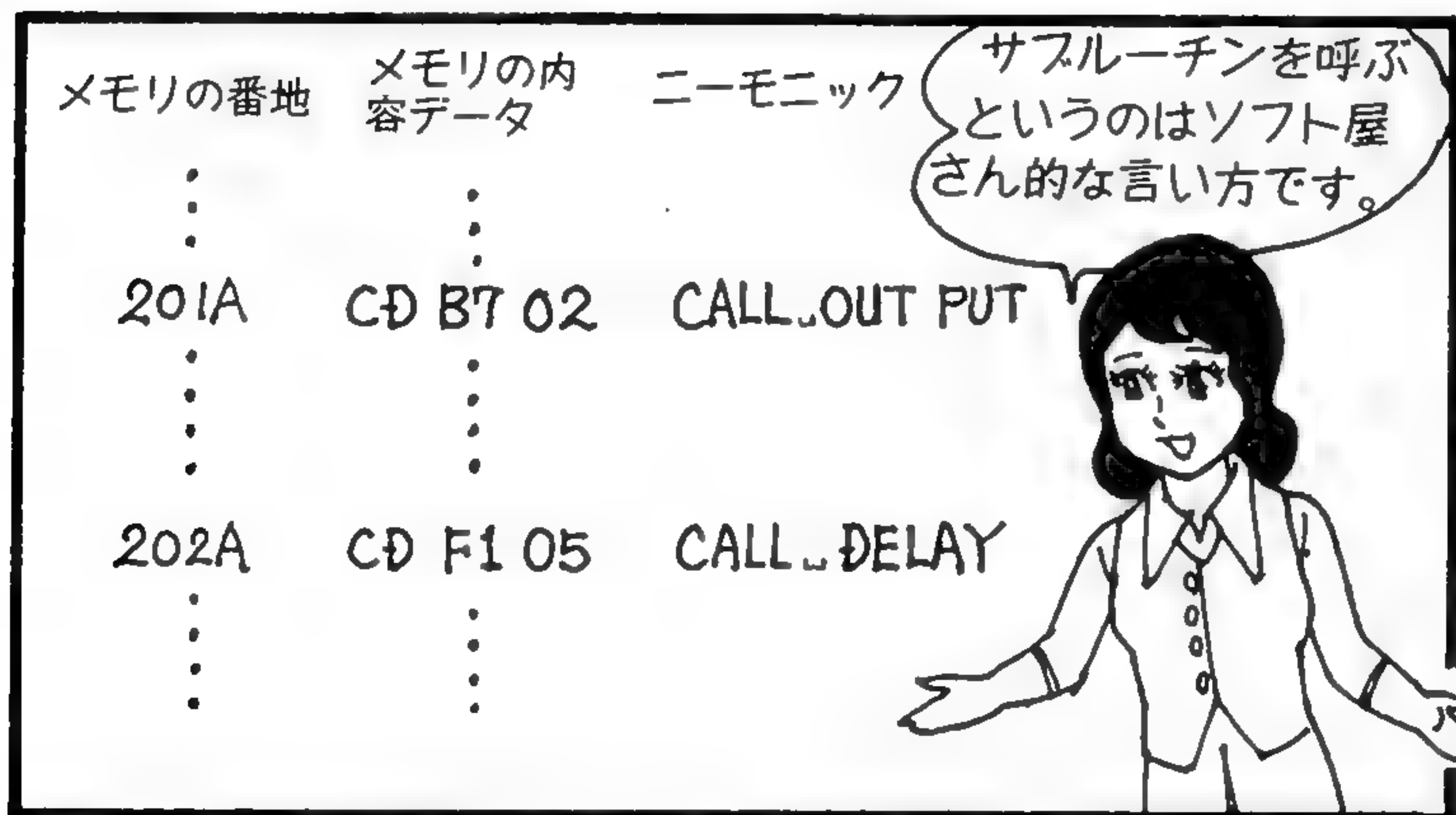
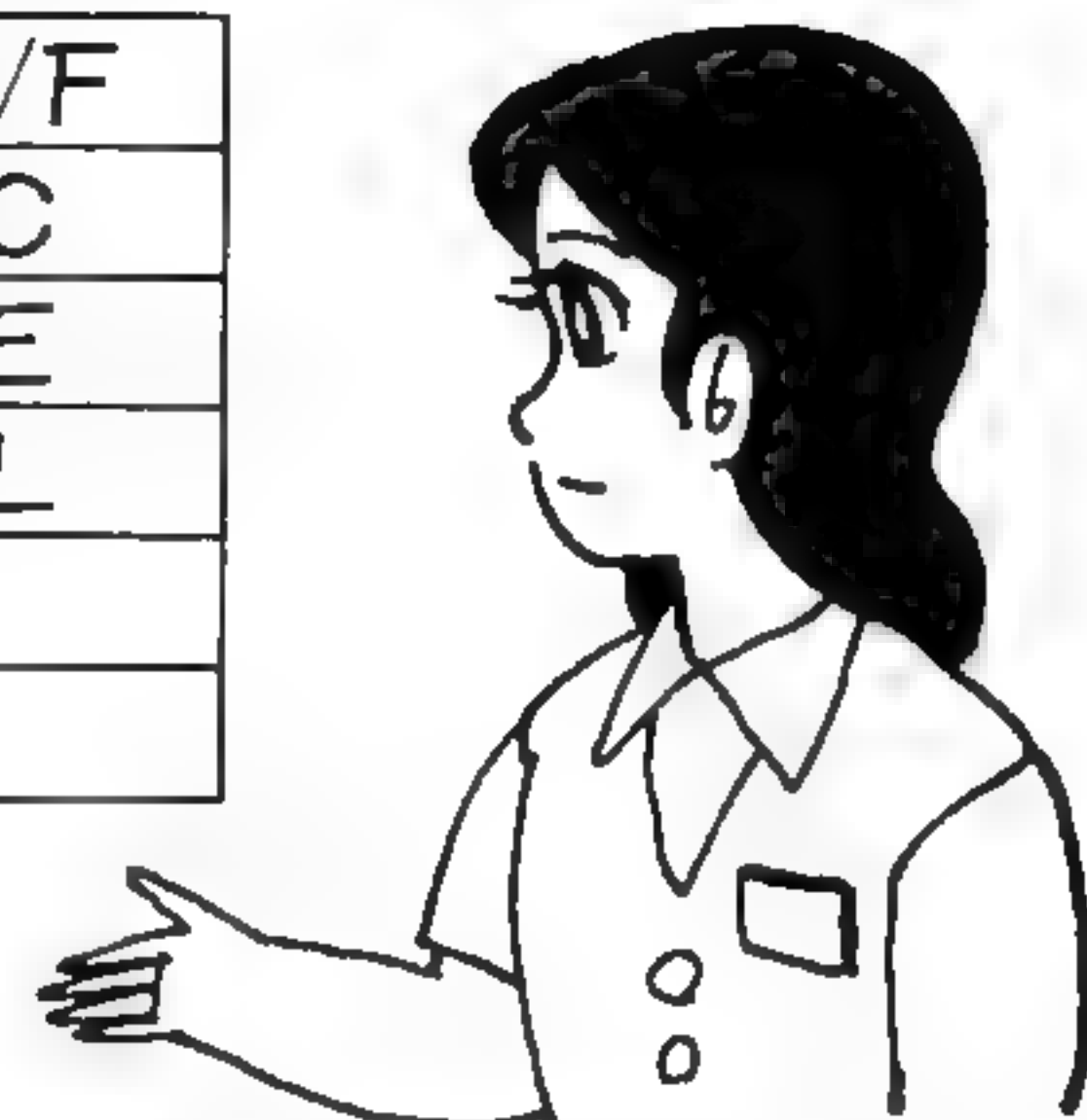




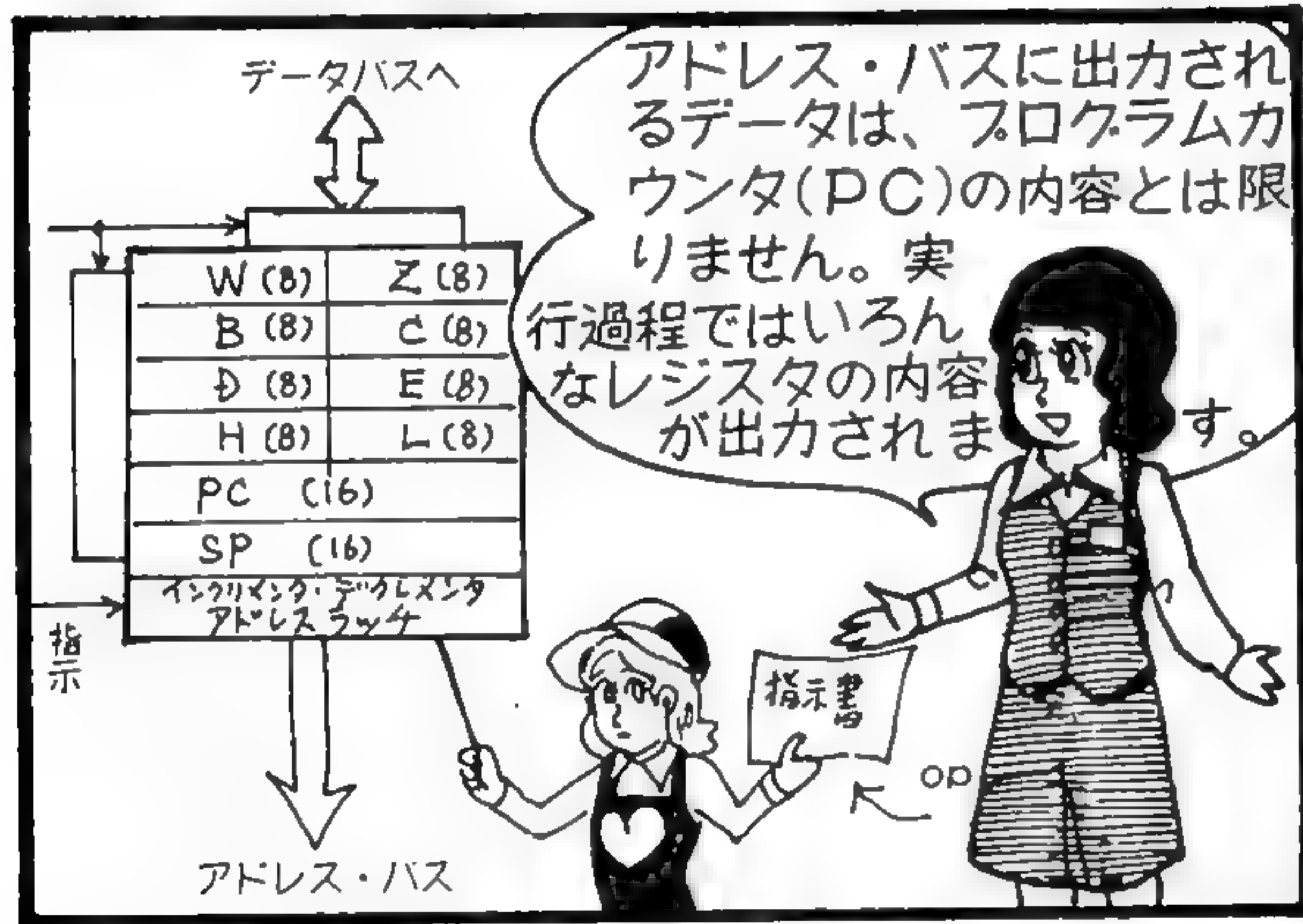
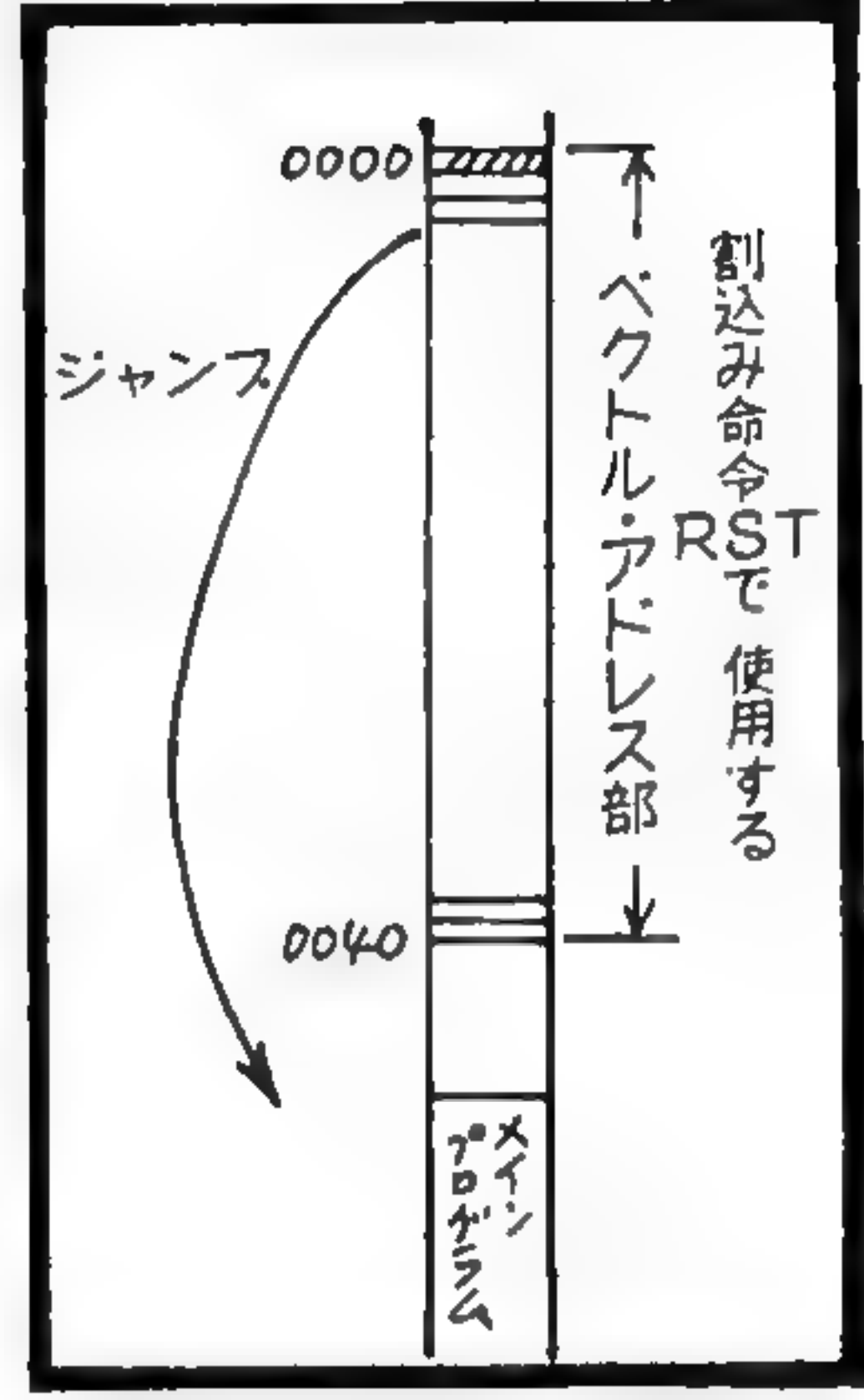


## ⑨プログラムカウンタとスタックポインタ

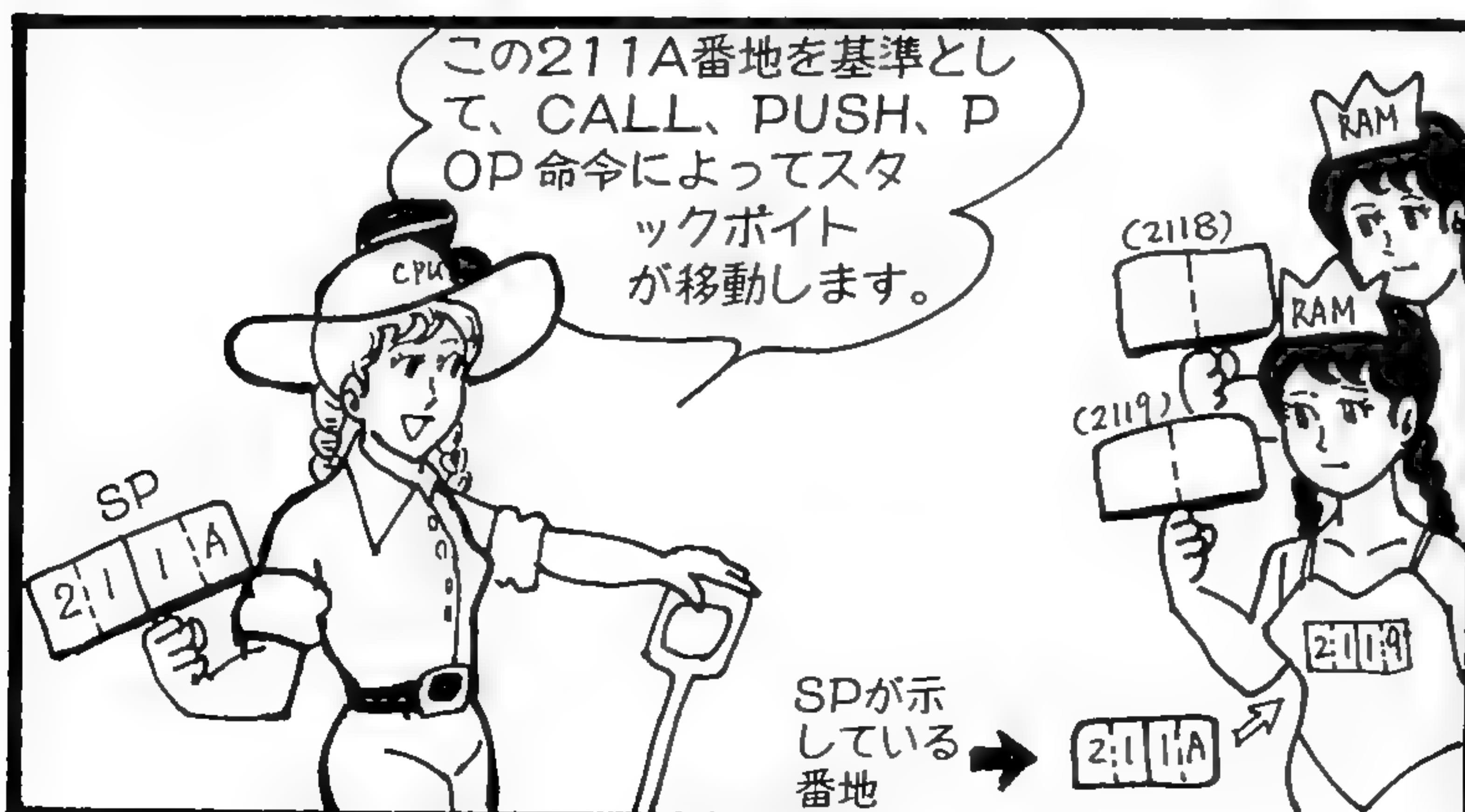
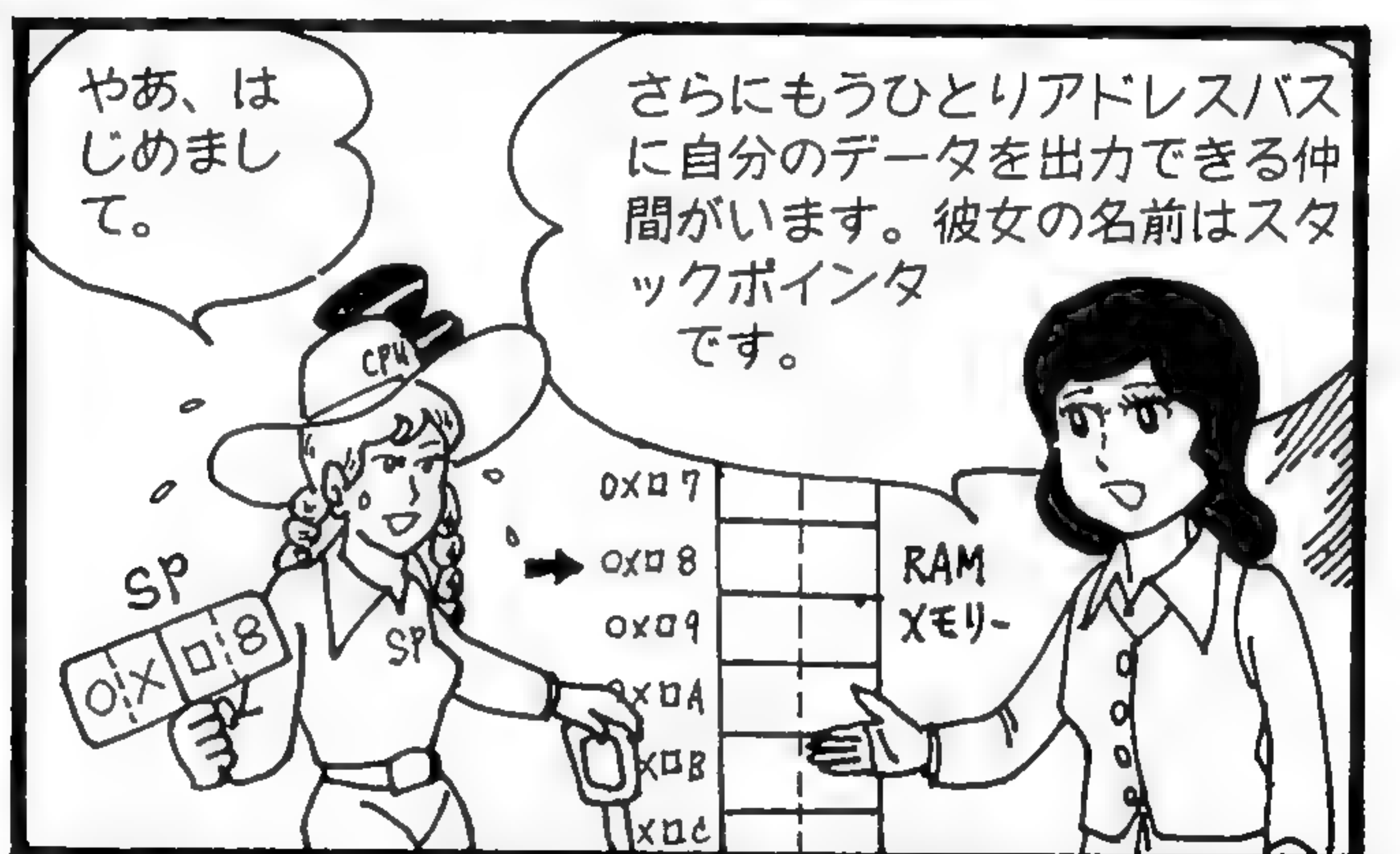
A	F/F
B	C
D	E
H	L
PC	
SP	



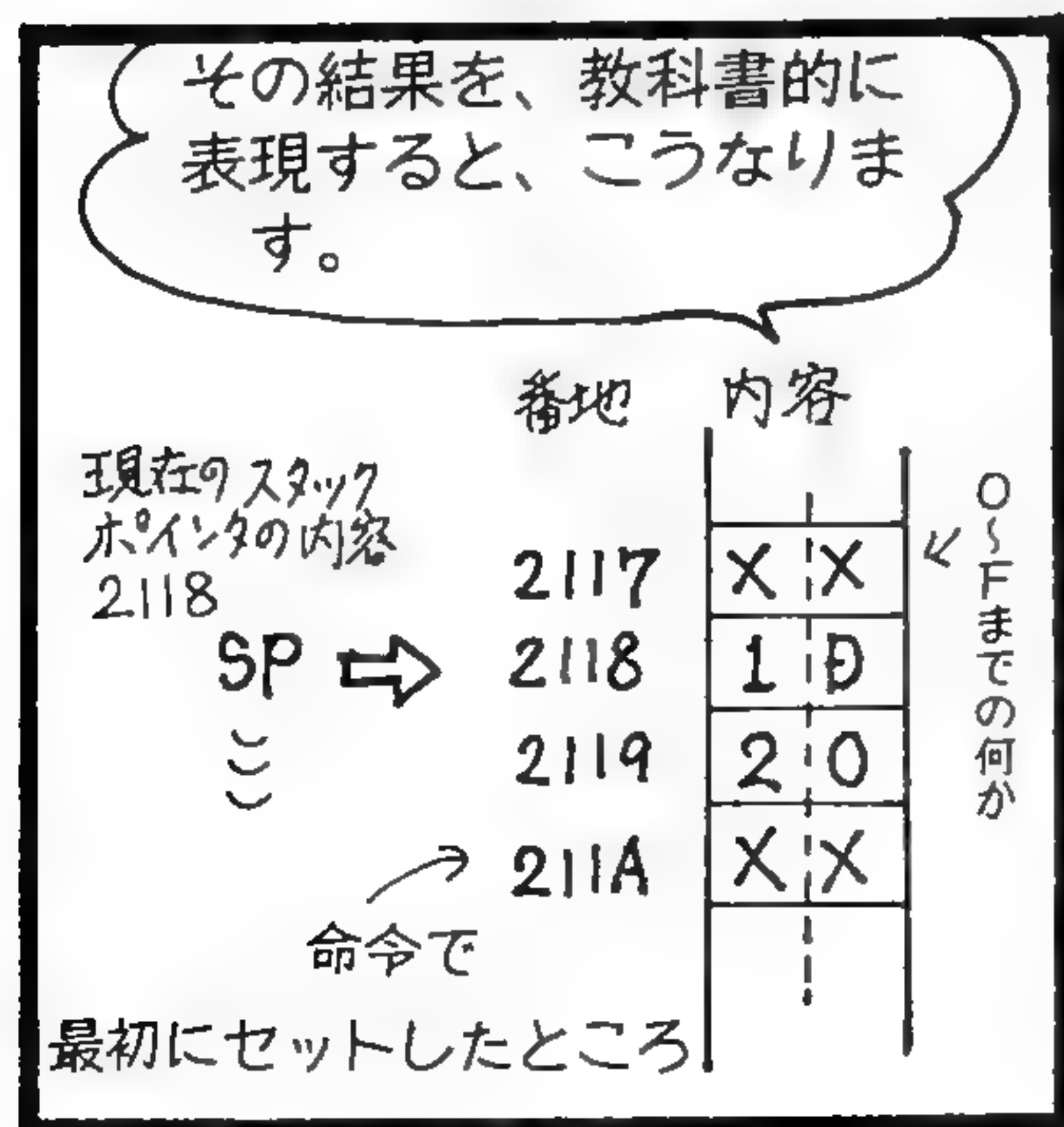
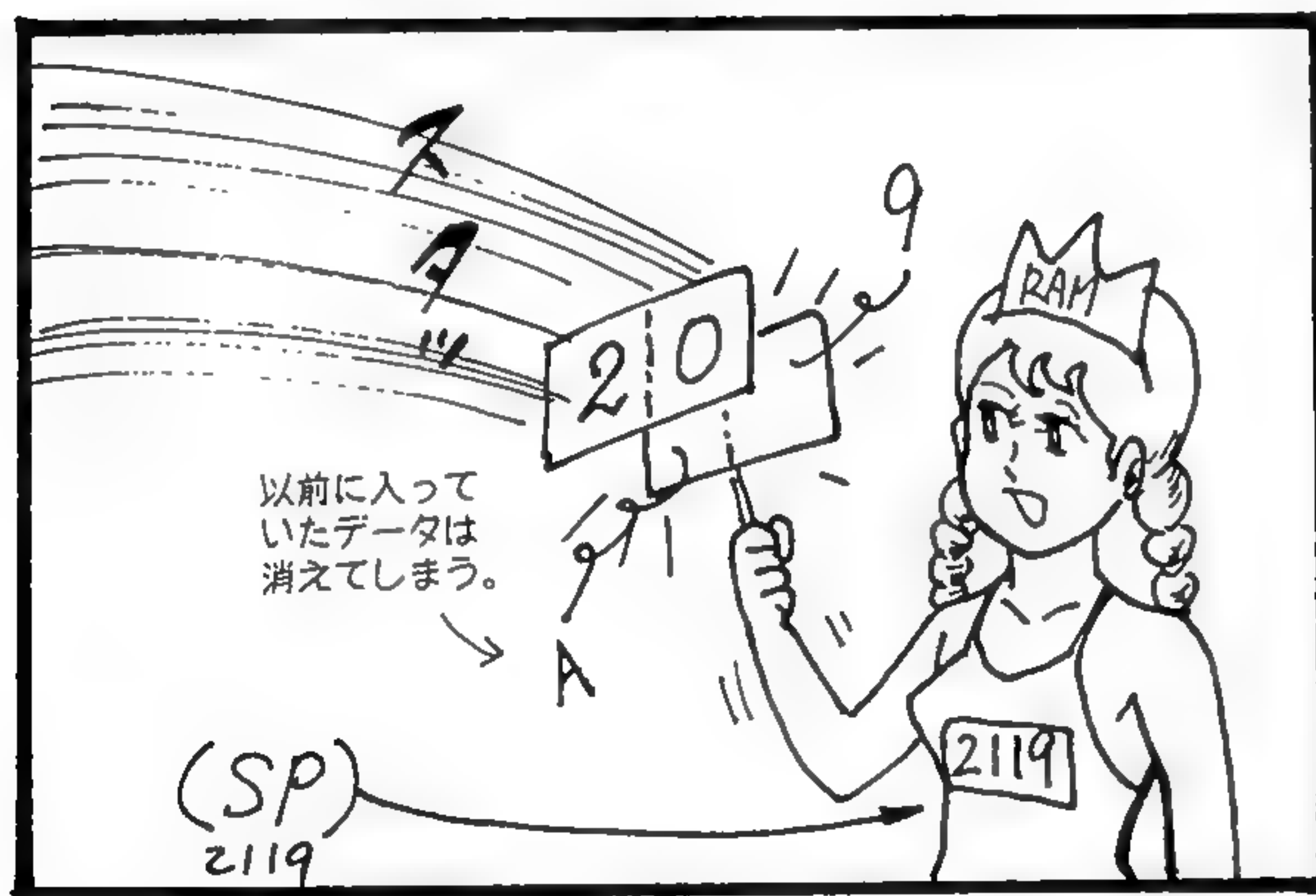
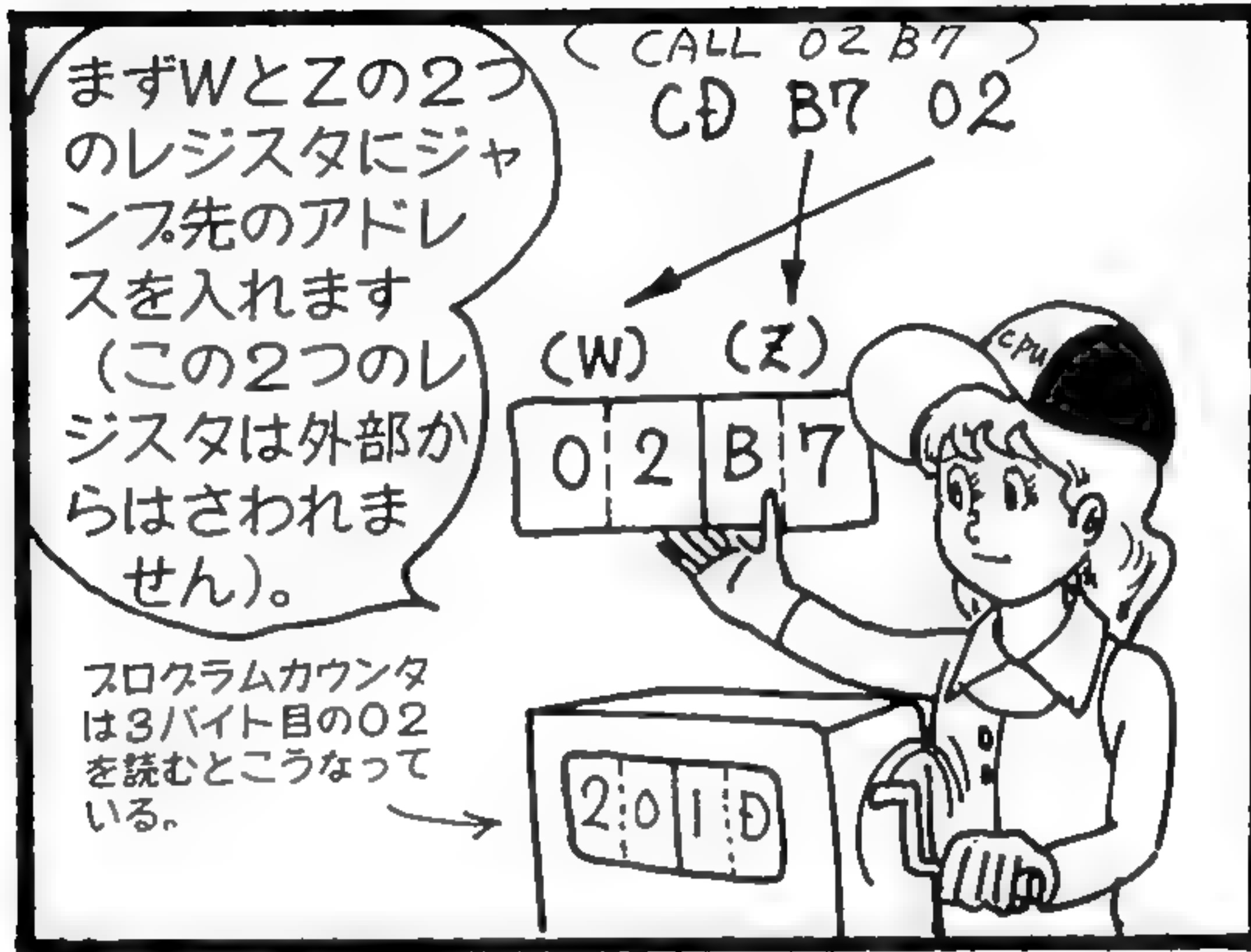










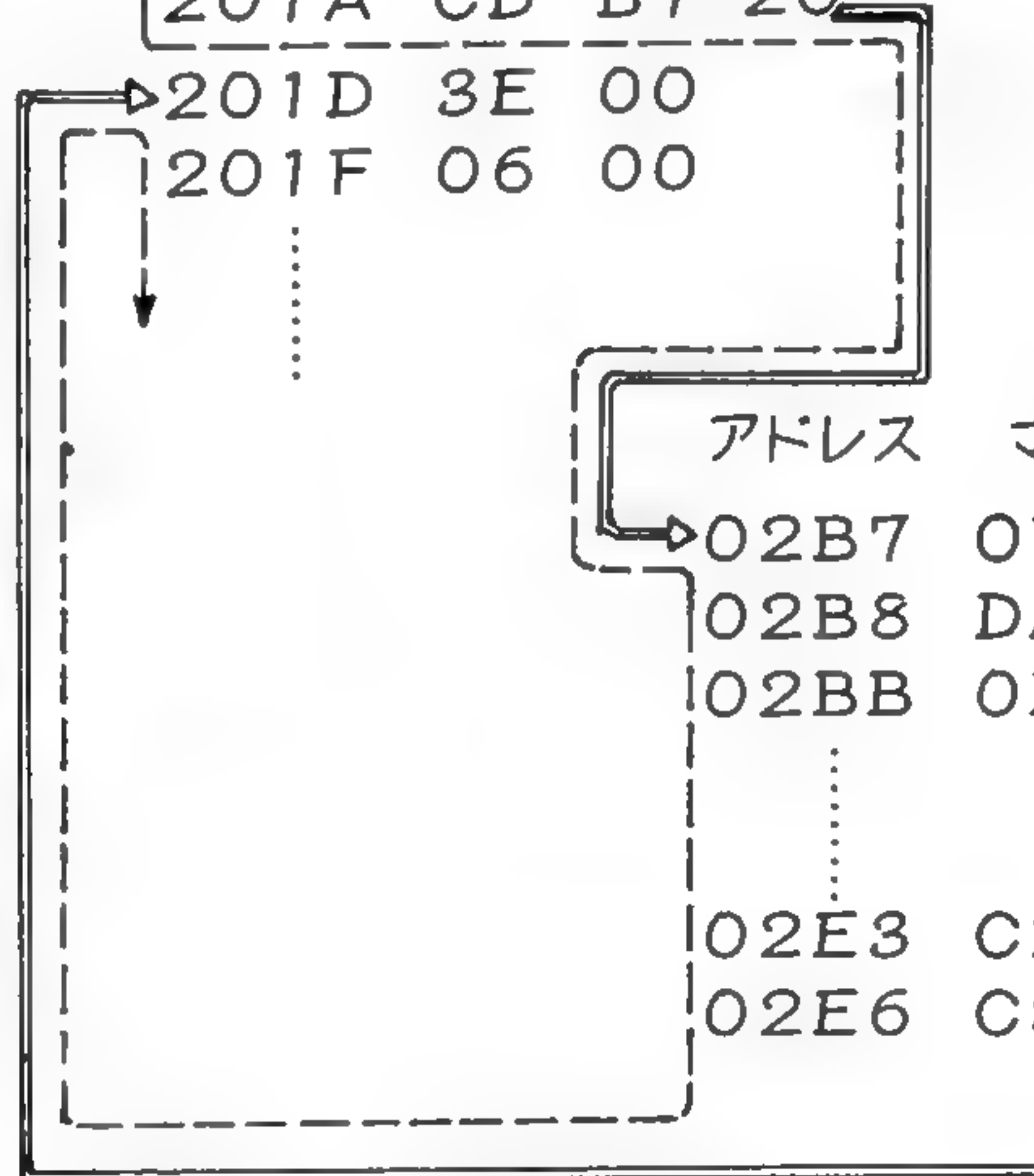






‘HELP’点滅プログラム(メイン)

ニーモニック	アドレス	マシン語
LXI SP, 20C8H	2010	31 C8 20
MVI A, 1	2013	3E 01
MVI B, 0	2015	06 00
LXI H, 2004H	2017	21 04 20
CAL OUTPUT	201A	CD B7 20
DPY: MVI A, 0	201D	3E 00
MVI B, 0	201F	06 00

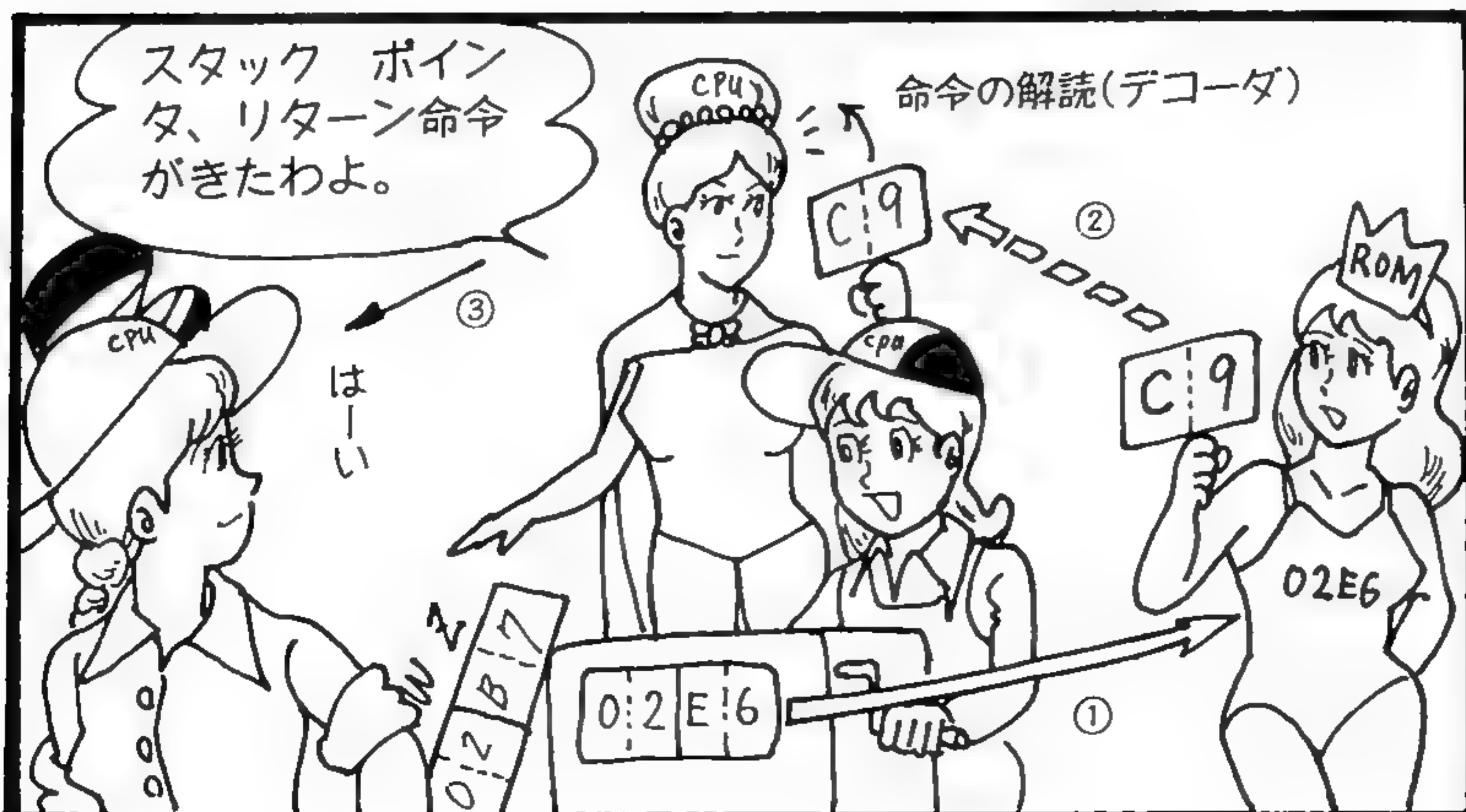


ワンボードマイコンではこのOUTPUTの他にDELAY(時間調整)ルーチンなどをモニタの中に持っていてユーザーが使えるようになっています。

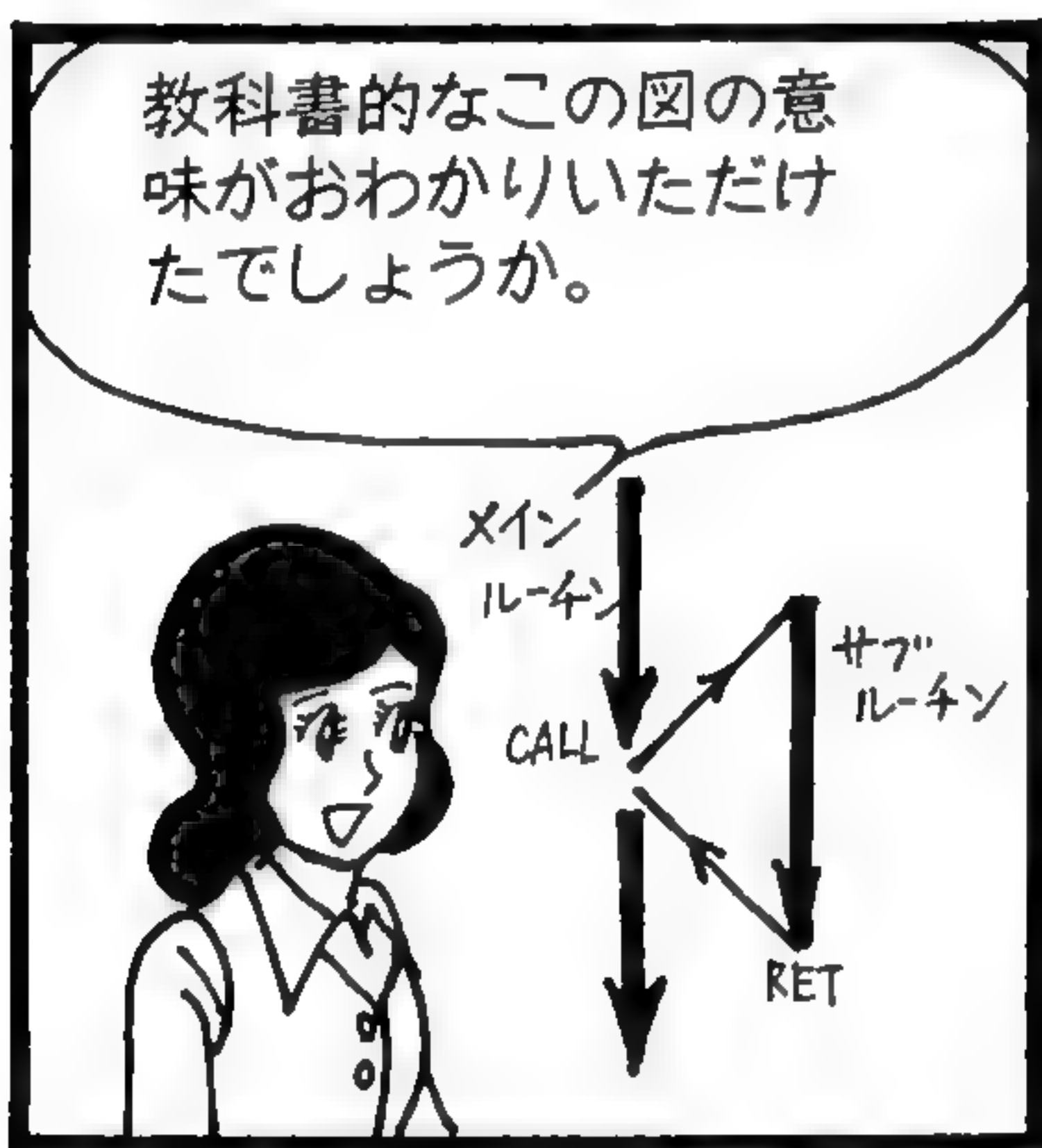
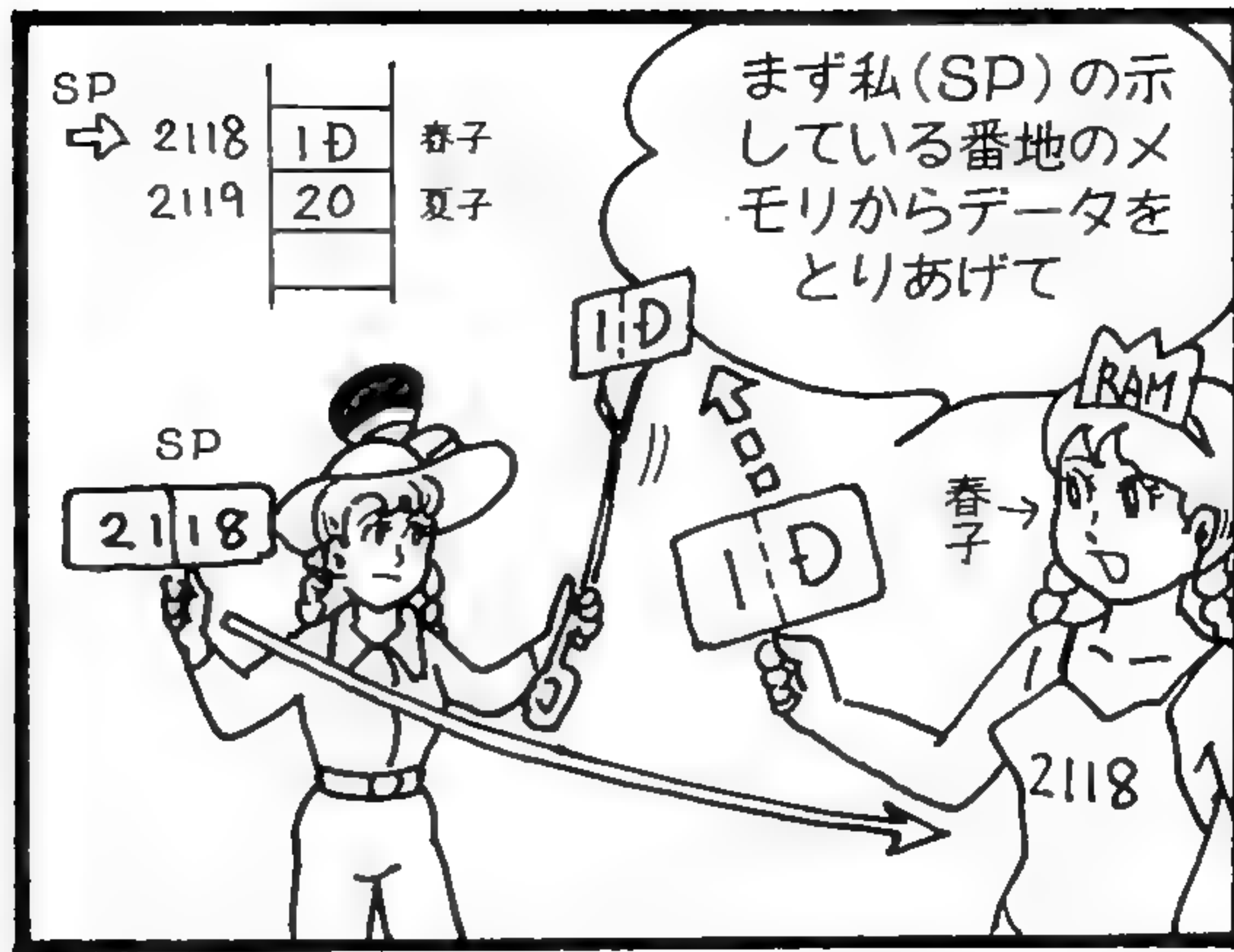
表示ルーチン(サブルーチン)

アドレス	マシン語	ニーモニック
02B7	0F	OUTPUT: RRC
02B8	DA C2 02	JC OUT05
02BB	0E 04	MVI C, 4
...	...	...
02E3	C2 C9 02	JNZ OUT15
02E6	C9	RET

-----> はプログラムカウンタのマラソンコース













そして、POP B命令によってRAMのデータがBCレジスタに戻されます。

① SP=2118  
② SP+1=2119  
③ SP+1=211A

RAMの中に、その内容が保持され、番地はSPが記憶しています。

SP → 2118 76 (C)  
2119 54 (B)  
211A

このあと、CALL命令などで、サブルーチンを実行してレジスタB・Cの内容が変わっても、

MOV B,M  
MOV B,A

このようにスタックエリアにメモリされていきます。

① PUSH B  
↓  
② PUSH PSW  
↓  
③ PUSH D  
↓  
④ CALL DATE  
↓  
⑤ ...

たとえばこのようなプログラムでは、

一度書き込まれた内容はそのまま残っています。

⑥ POP D  
↓  
⑦ POP PSW  
↓  
⑧ POP B  
↓  
...

戻す時は順序が逆になります。

スタックとして使えるメモリはRAMちゃんだけです。

だいたい8080の動作についてわかっていただけましたか？ RST、IN、OUTの3つの命令についてはハードウェアのところでくわしくご説明します。

レジスタ間でデータチェンジしたことになります。

PUSH B  
PUSH H  
POP B  
POP H

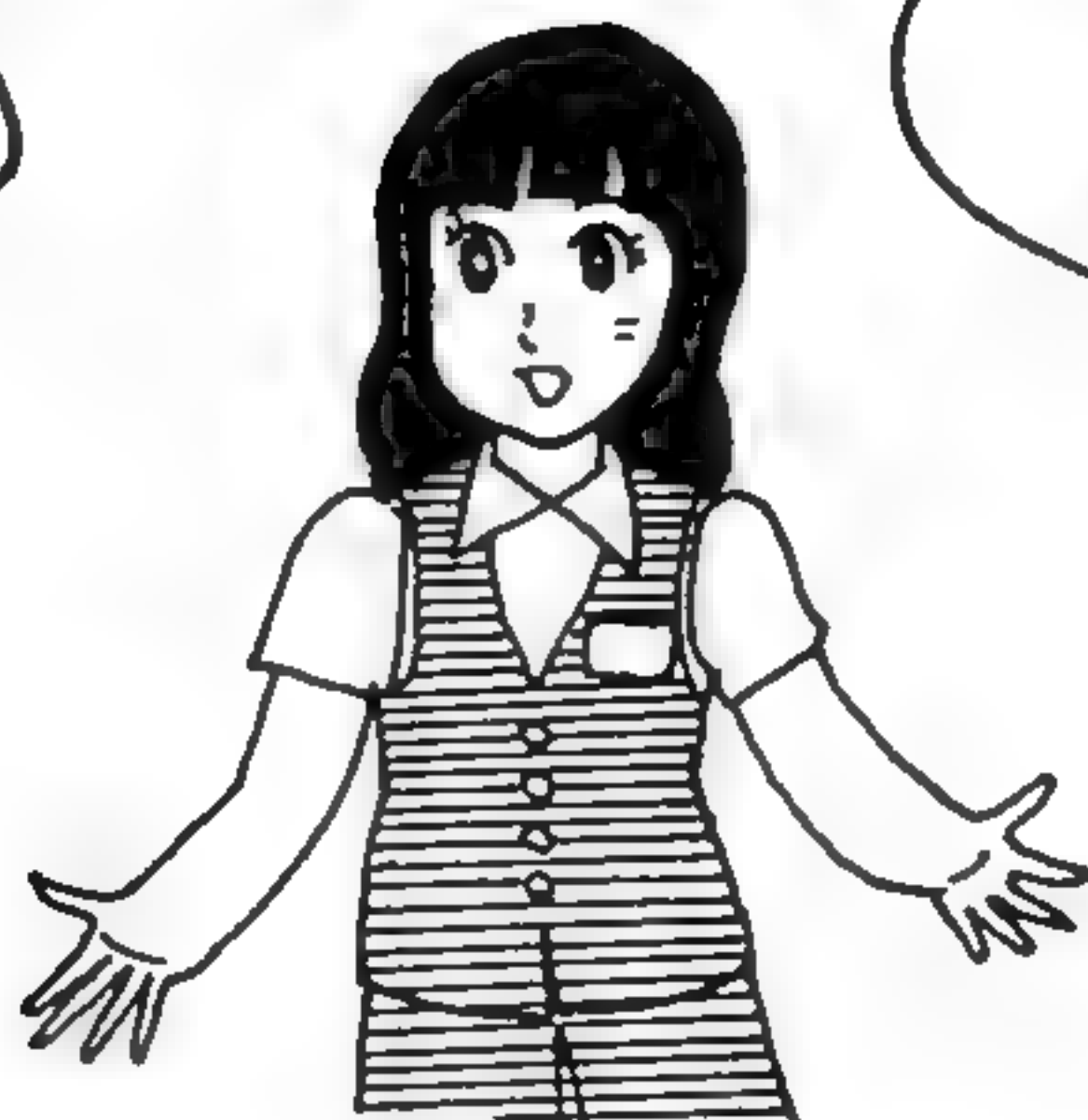
この時わざと戻す順序を変えれば、



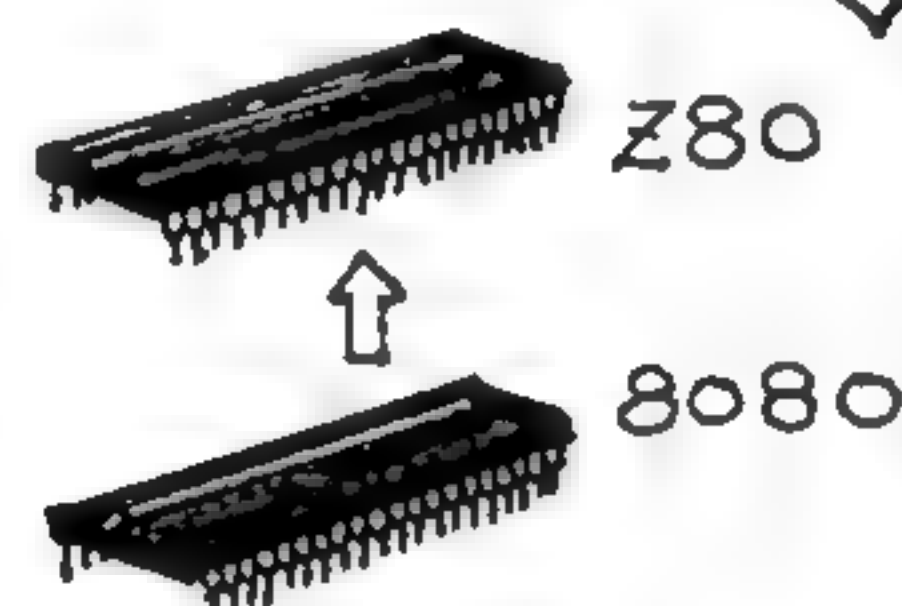
## ⑩ Z80

日本では 6809 も大きな勢力を持っていますので、簡単にこの2つのCPUの特徴をお話しましょう。

今8ビットCPUの主流は 8080 の基本ソフトウェアがそのまま使える Z80 です。



8080のことがわかっていると、少しのヒントでZ80もわかるはずですよ。マイクロコンピュータを自在に使いこなすには、CPUのことがわかってないとダメということは何度も痛感しています。



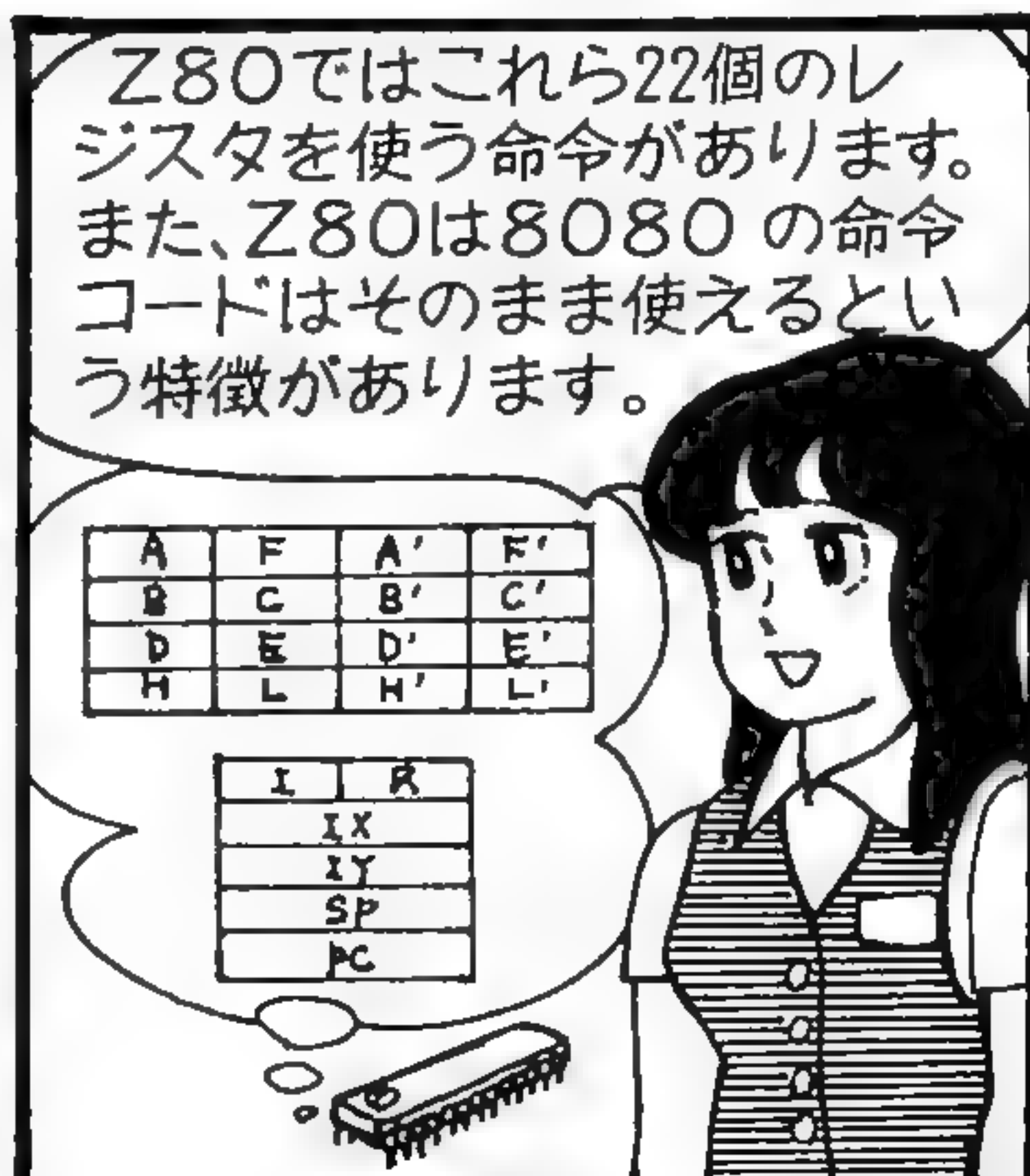
08  
10  
18  
28  
38  
CB  
D9  
DD  
ED  
FD

30と20は8085で使っている



命令コードのマップ

	0	1	2	3	...	E	F
0	NOP						RP
1							
2							
3							
...							
F	RRO						

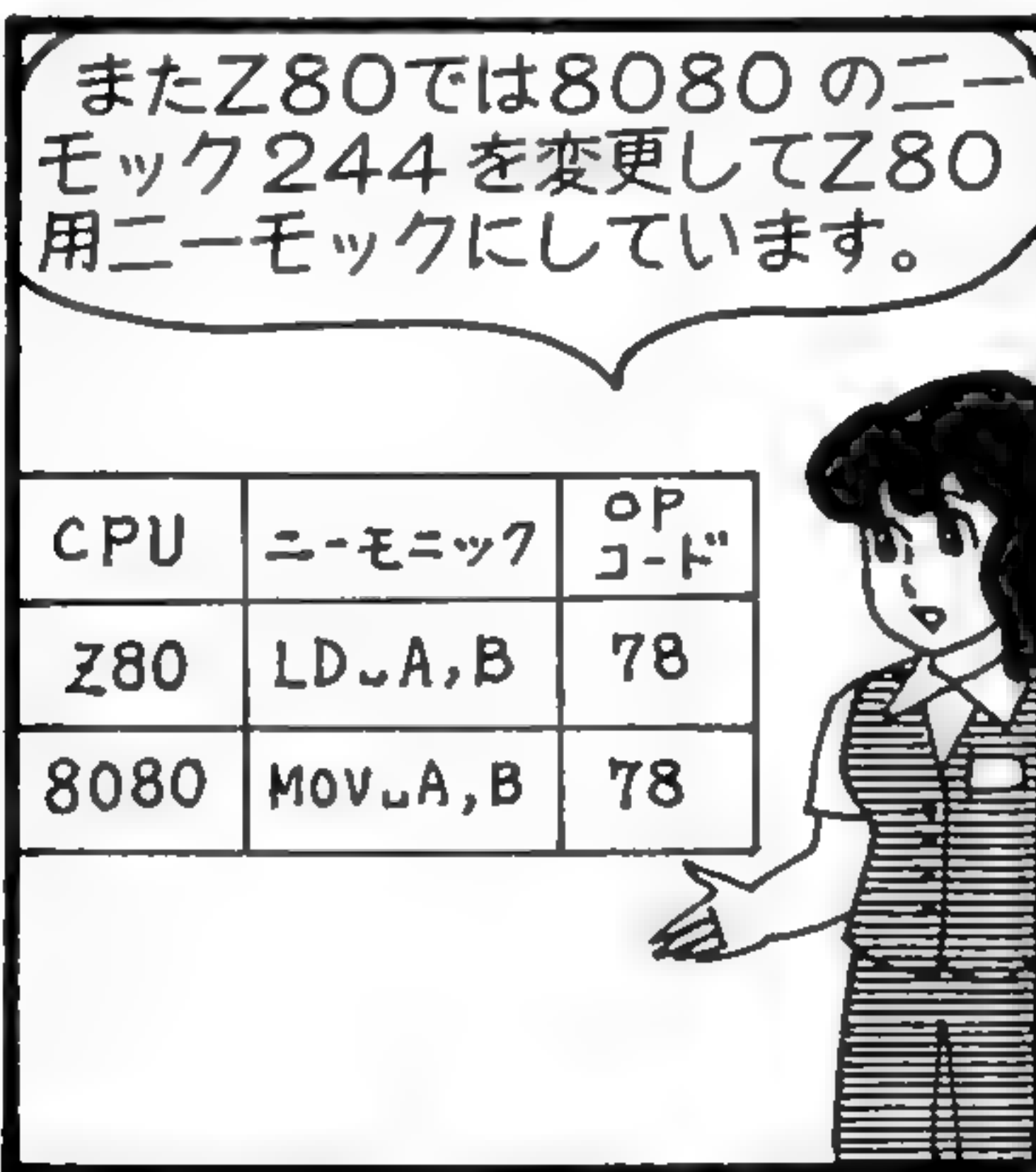
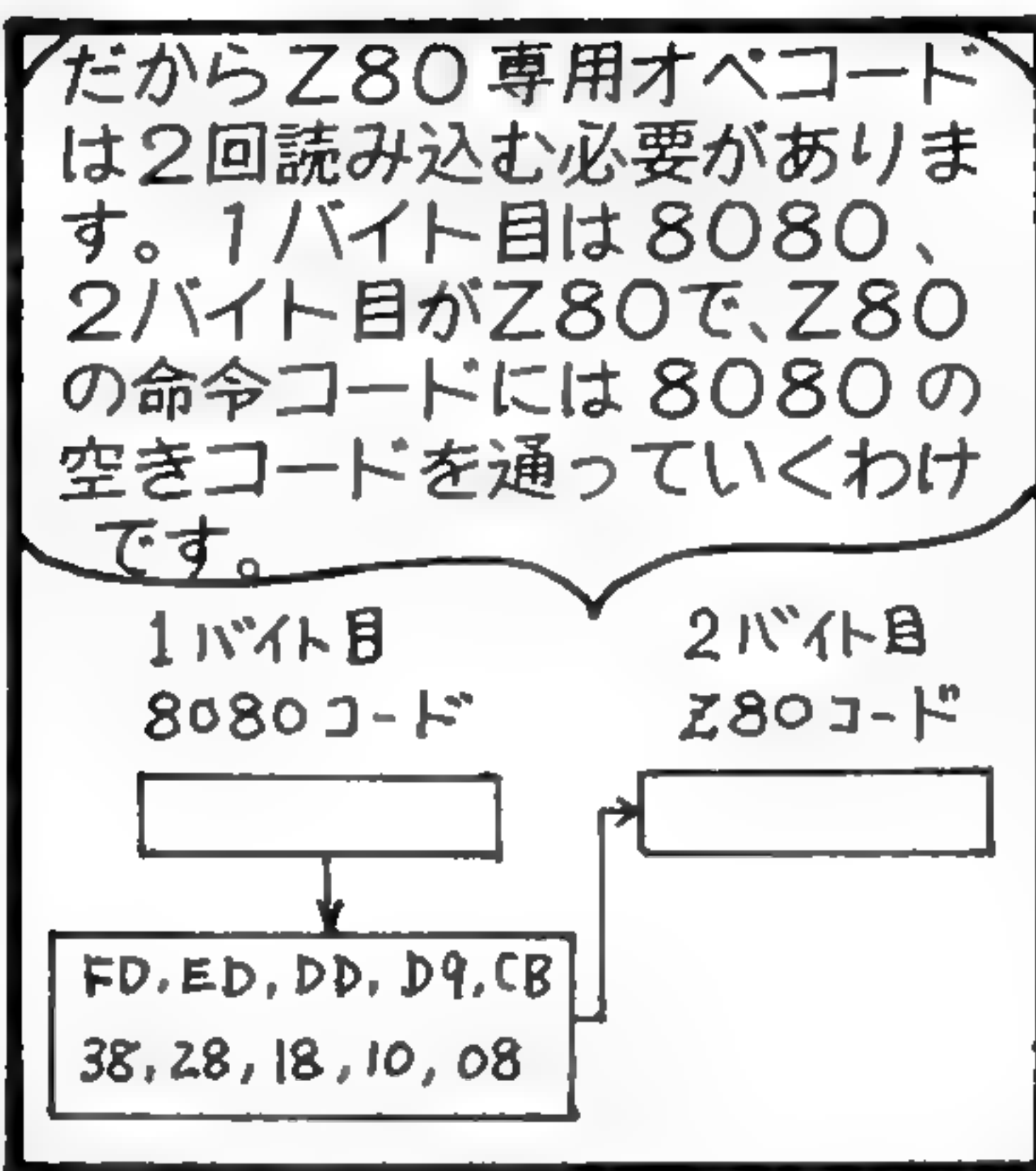
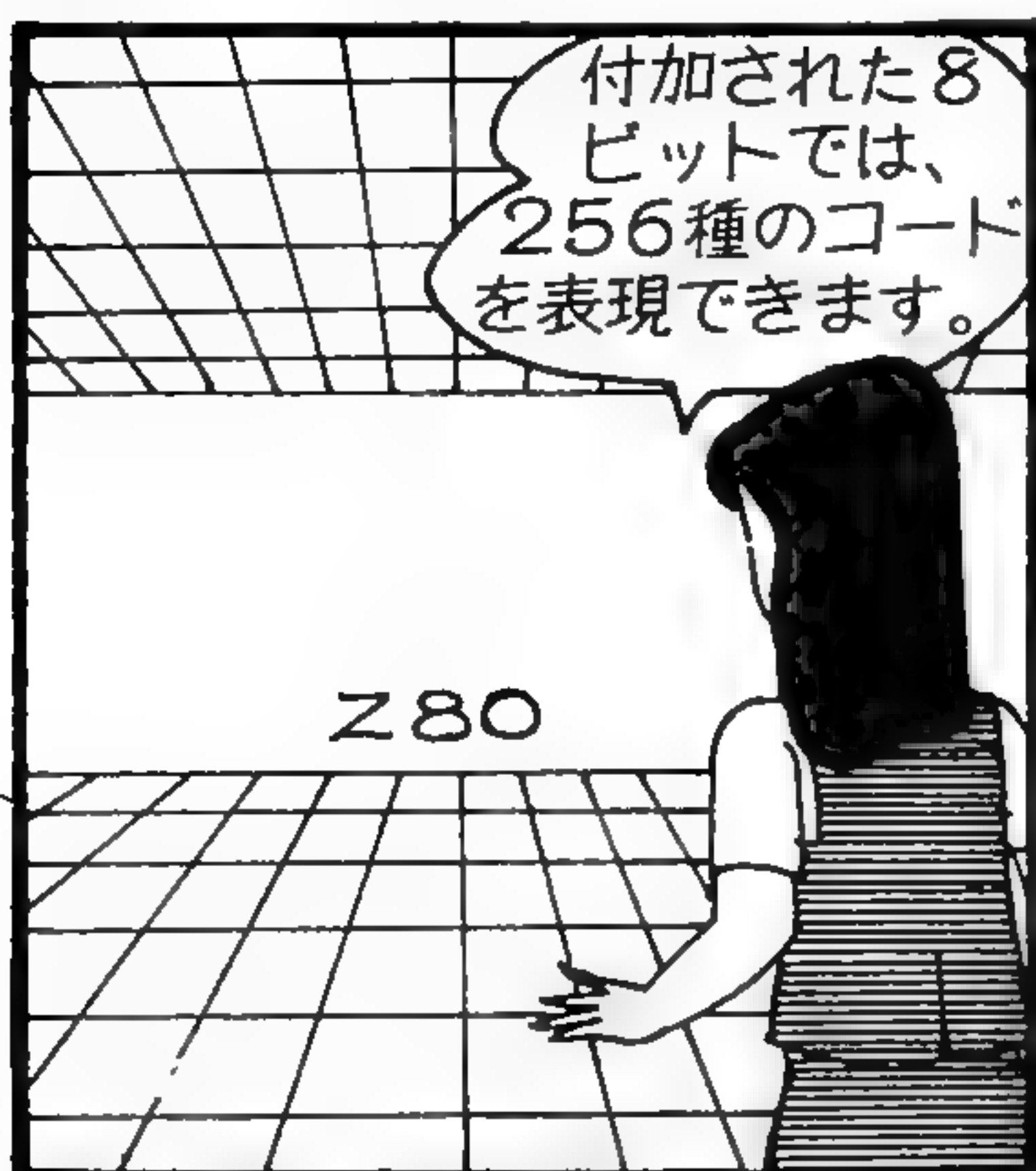


A	F	A'	F'
B	C	B'	C'
D	E	D'	E'
H	L	H'	L'

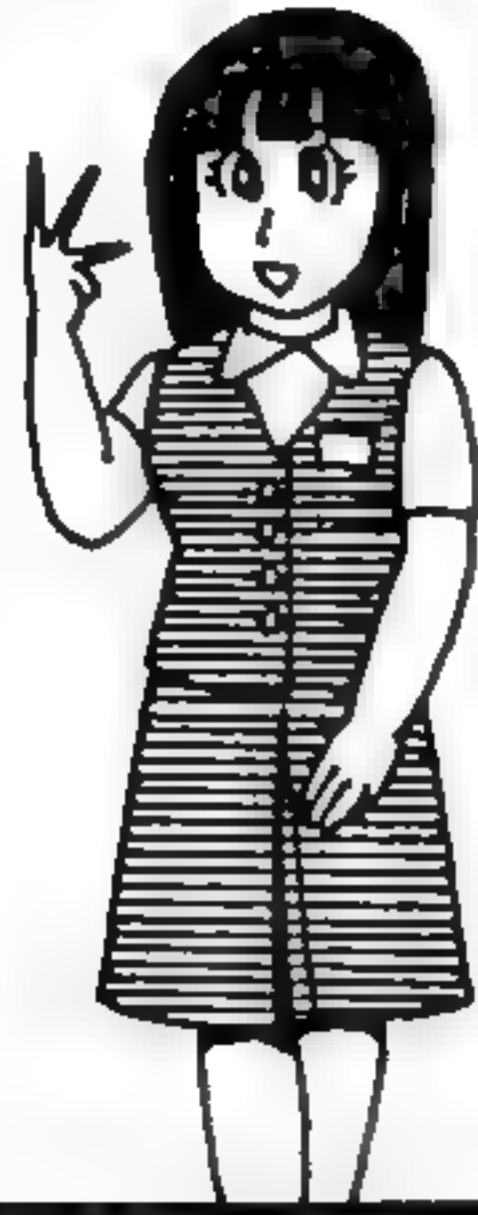
I	R
IX	
IY	
SP	
PC	







では、このうち08、20、D9の3つのOPコードの動作内容を説明します。



1バイトのOPコードでZ80の専用になったのは、この8つのコードです。

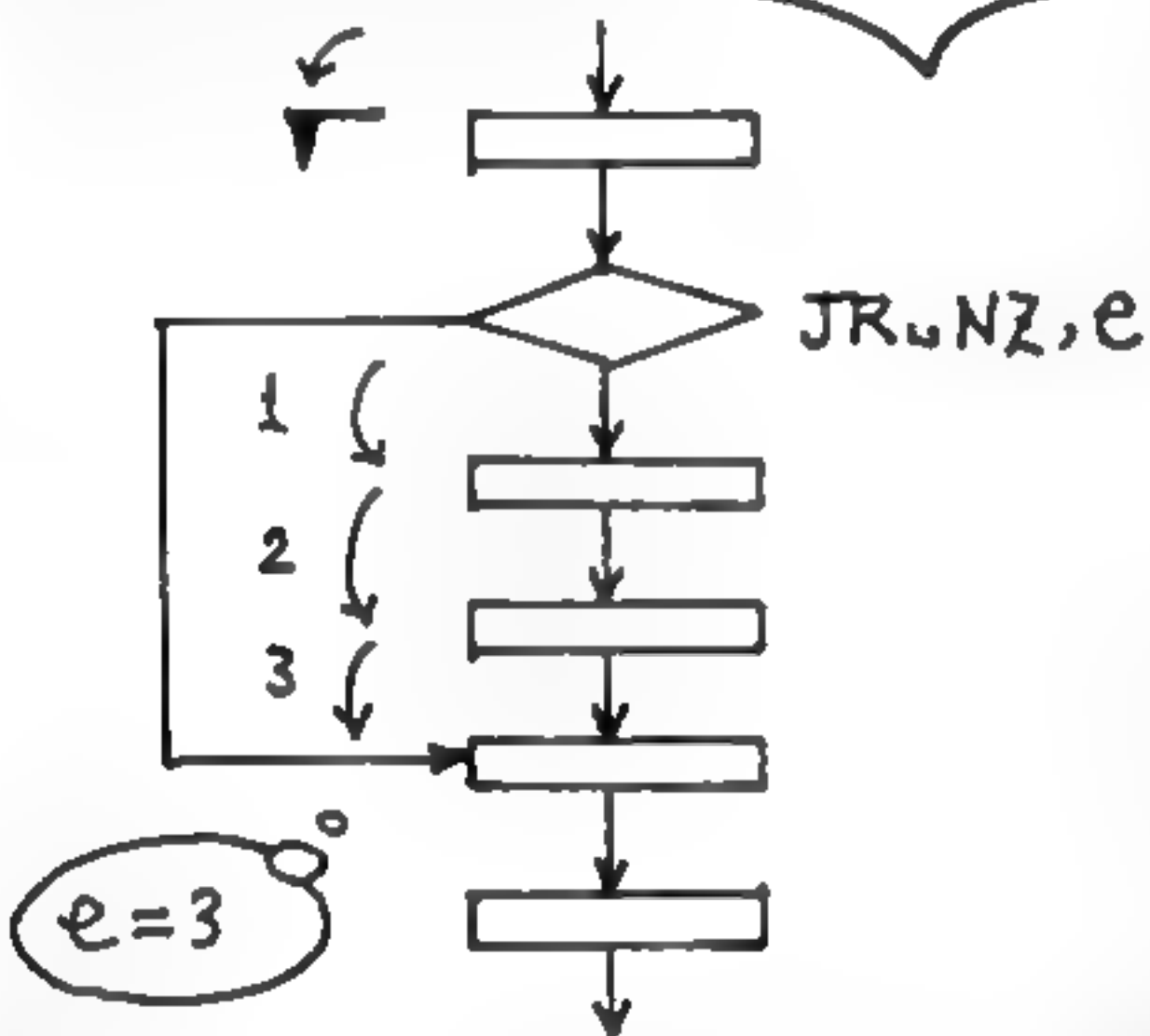
08 ... EX<sub>h</sub>AF, AF'  
10 ... DJNZ, e  
18 ... JR<sub>h</sub>e  
20 ... JR<sub>h</sub>NZ, e  
28 ... JR<sub>h</sub>Z, e  
30 ... JR<sub>h</sub>NC, e  
38 ... JR<sub>h</sub>C, e  
D9 ... EXX



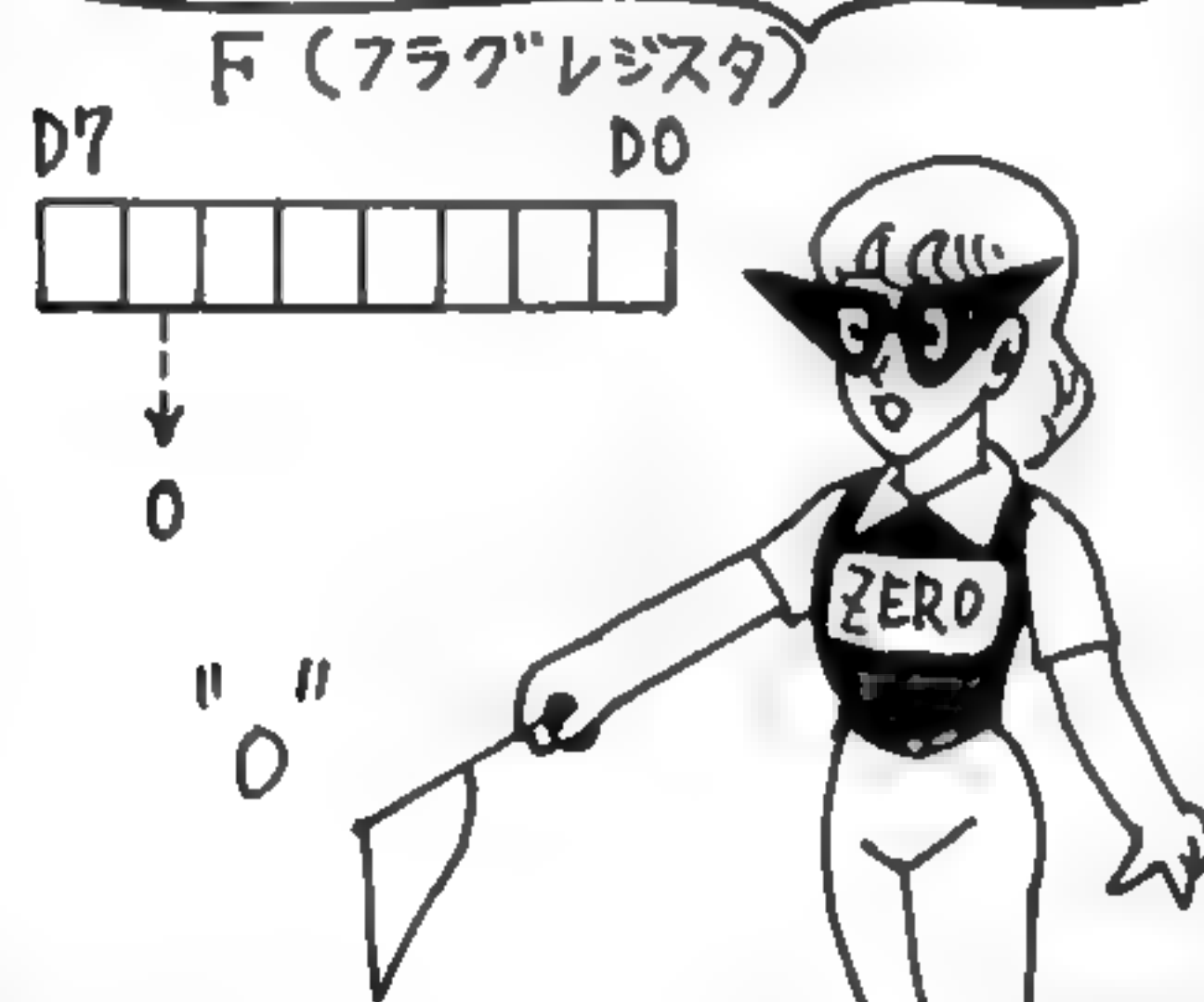
まず8080との共通コード以外のものから見ていくことにします。



この命令の前にゼロフラグを变化させる命令をおき、結果が'0'ならばジャンプさせるようにします。



20...JR<sub>h</sub>NZ, e これは相対ジャンプ命令です。ジャンプの条件はゼロフラグが'0'のときです。



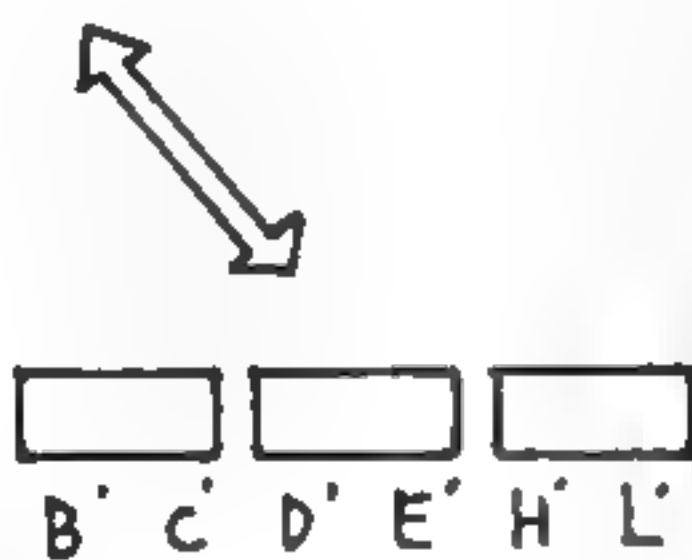
08...EX<sub>h</sub>AF, AF' これはレジスタ同士の内容交換です。(ダッシュ)付きレジスタはこの交換以外の役目はありません。



D9...EXXはBC、DE、HLとBC'、DE'、HL'間でデータの入れ換えをします。CALL文の直前に置いてデータを保護します。

B C D E H L

POP・PUSHを使うより簡単にできる。

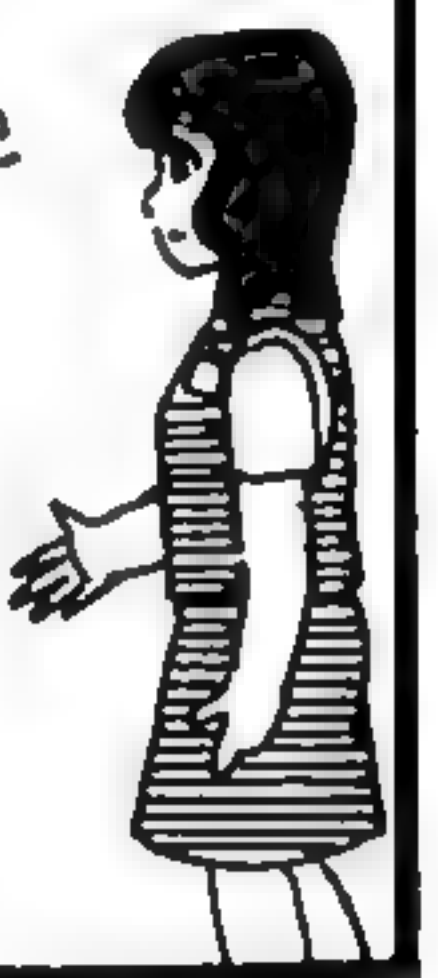
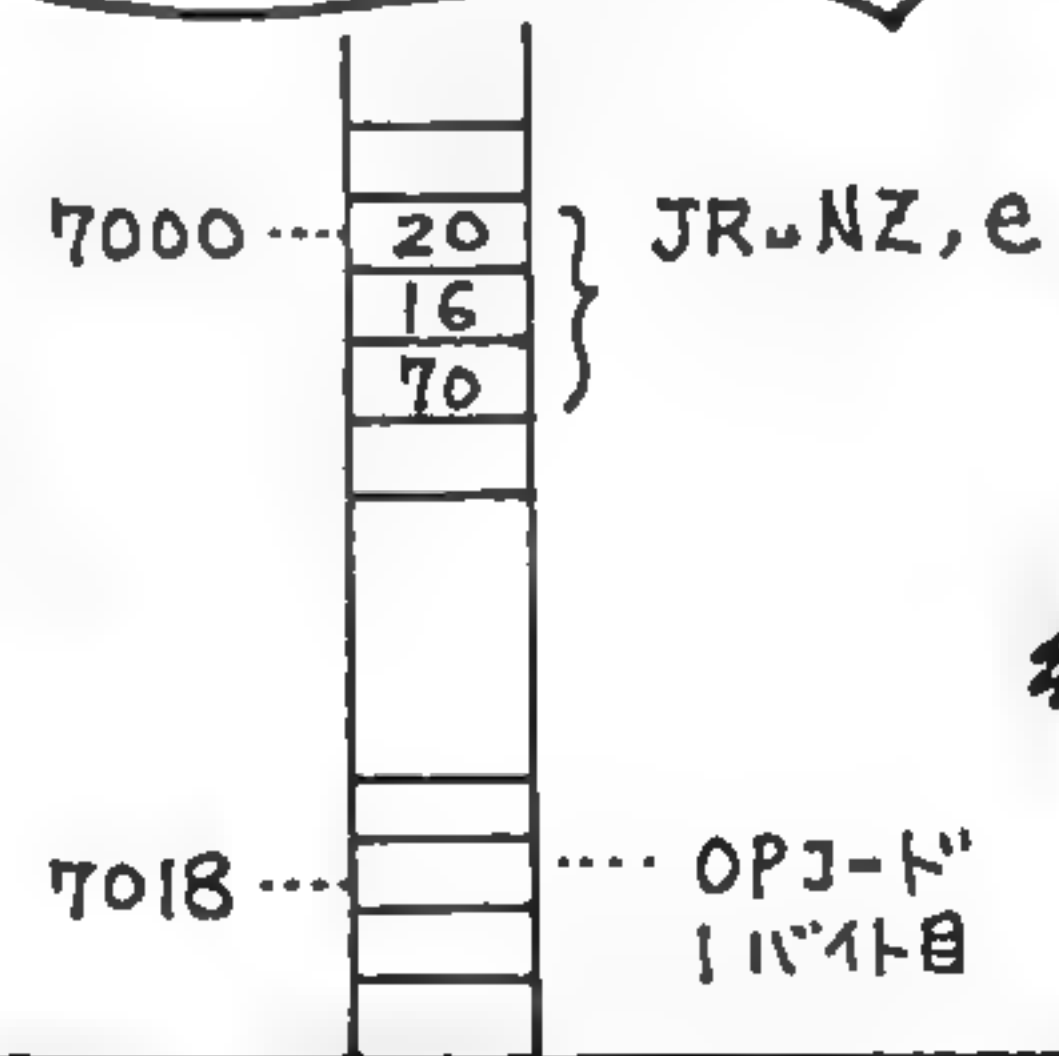


ただ、命令の中に直接このeを書き込むことはできません。eだけ離れた番地を書くか、モニタックならラベルを使います。

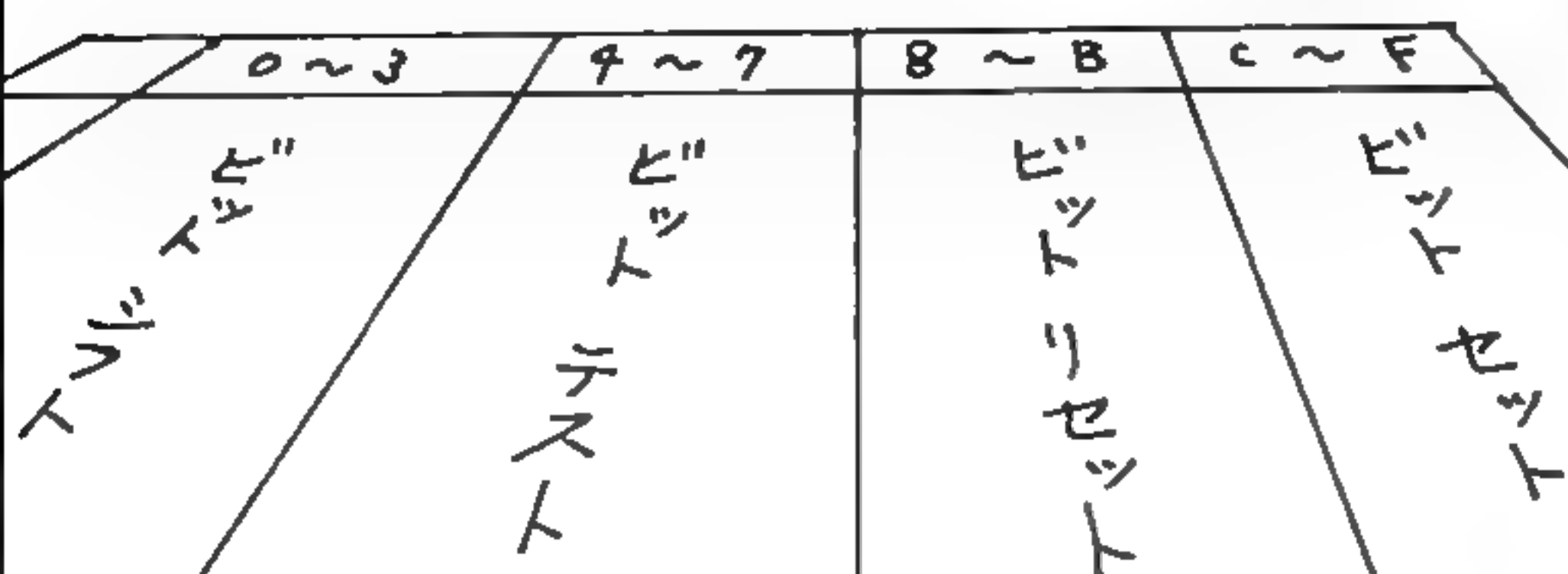
ラベル表記  
↓  
JR<sub>h</sub>NZ, DONE  
直接数値  
↓  
7000 20 16 70  
:  
7018



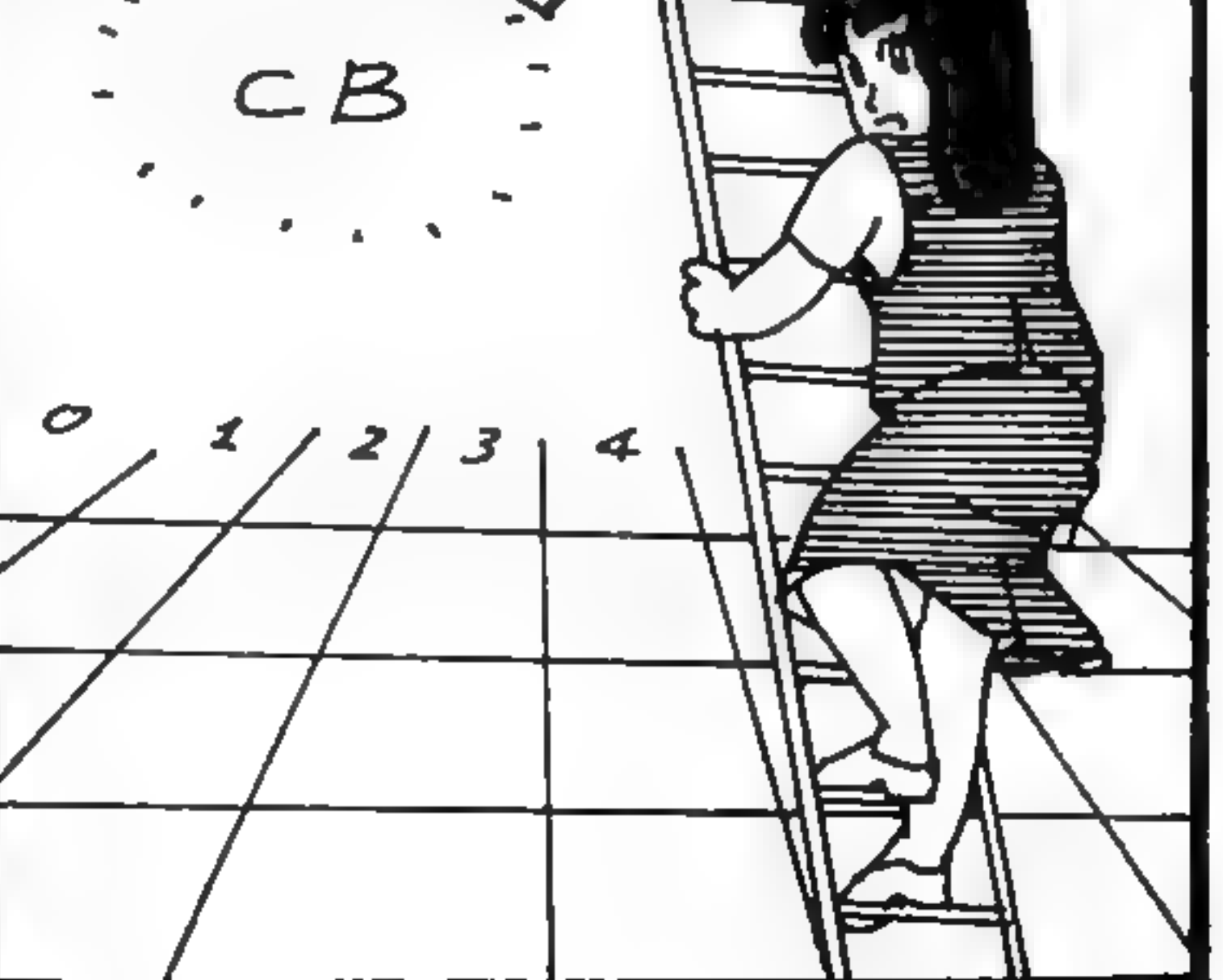
また、eというのはJR<sub>h</sub>NZ, eのOPコード20の書かれている番地からの相対的な距離を示しています。



まずCBを1バイト目にもつ命令群を見てみましょう。



残りのCB、DD、ED、FDの4つのコードが2バイト、3バイトのOPコードへの入口になっています。





I/Oポートについてはこの後の8080のハードウェアの所で詳しく説明しています。

メモリのアドレスに相当する↓

DEV. No.	D7	D6
00	印字機	
01		

このビット操作命令はI/Oポートのハンドリングをわかりやすくします。

このCB...で表現されるOPコードはビット操作命令です。

チェック  
リセット  
セット

ビットチェック命令を使うとこうなるの。ニーモニックBIT 7, AによってAレジスタのD7のビットをチェックしようとしていることがわかります。

```

IN A, (00)
BIT 7, A
Z=1 JP Z, n1 n2
Z=0
    
```

このZ=1というのがくせものだったわね。結果が0のときがゼロフラグが'1'だけセットされるわけ。

Zフラグ="1"

A 0000 0000

ゼロ!

これがそのチェックルーチンです。

```

IN A, 00
ANI A, 80
Z=1
JZ
Z=0
    
```

BIT 7, AはAレジスタのD7のビットをチェックするものです。

D7=1だからZフラグをおろす

D7, D6 D5 D4

1

IN A, (00)の内容です。

00ポートのデータをAレジスタに入れます。

こうしてみるとZ80のニーモニックの方が命令の動作内容をよく表現しているように思えます。

Z80 8080

IN A, (00) IN A, 00

OPコードはどちらもDBです。

(HL)は前にでてきた(M)に当たります。つまり( )内は番地ポートNo.を示し( )を含めるとそれは示された番地やポートにあるデータ、ということになります。

H L

アドレス

BIT 1, (HL)

ところでIN A, (00)の( )表現はほかにもBIT 0, (HL)のように使われていますが、これはどのような意味があるのでしょうか。

HL と (HL)

SET(セット)、RES(リセット)命令もレジスタの中の任意のビットをON、OFFするもので、OUT命令と組み合わせて、たとえばパイロットランプ、リレーのON、OFFに応用できます。

SET 3, A

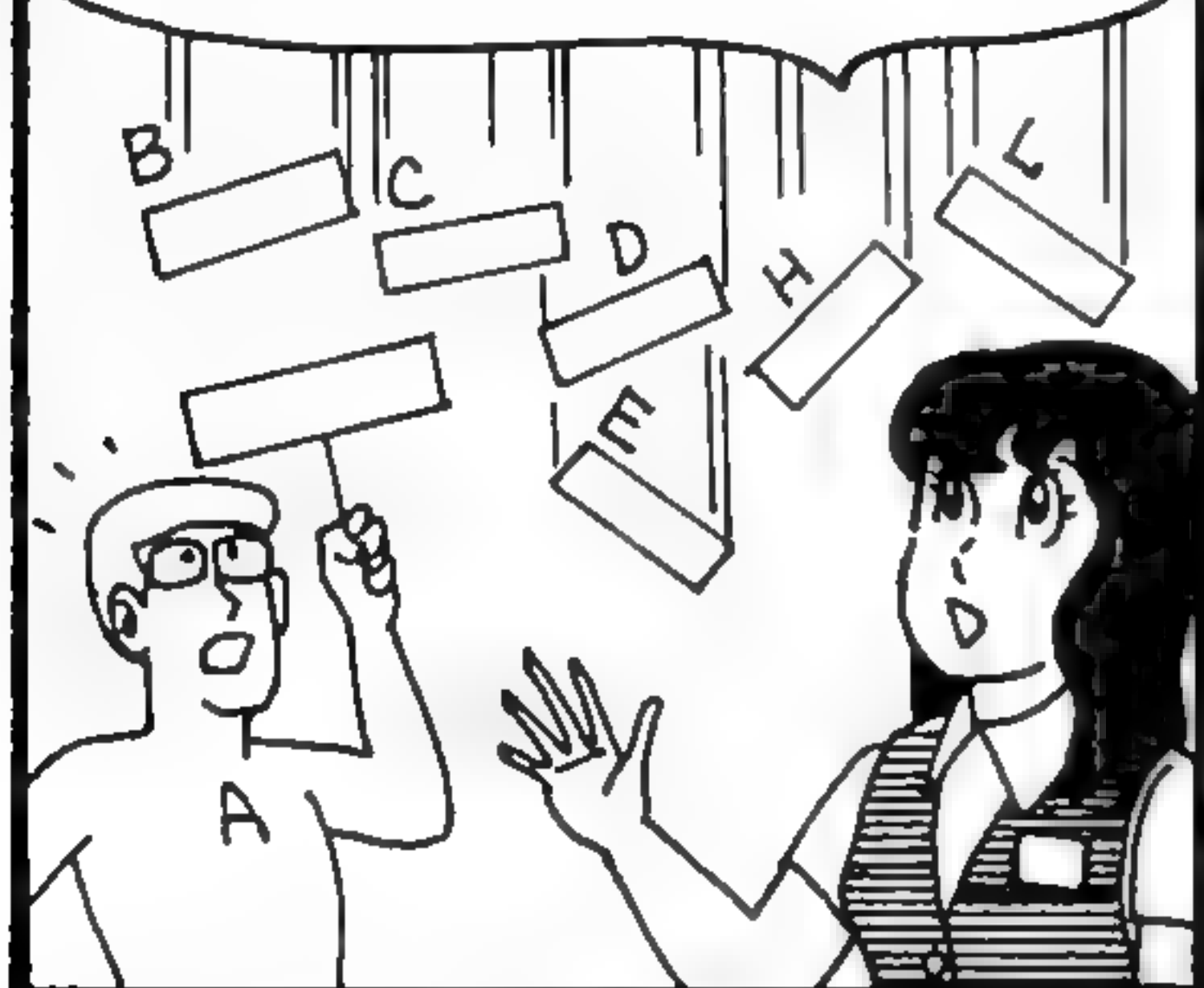
OUT (01), A

A D7 ... D0

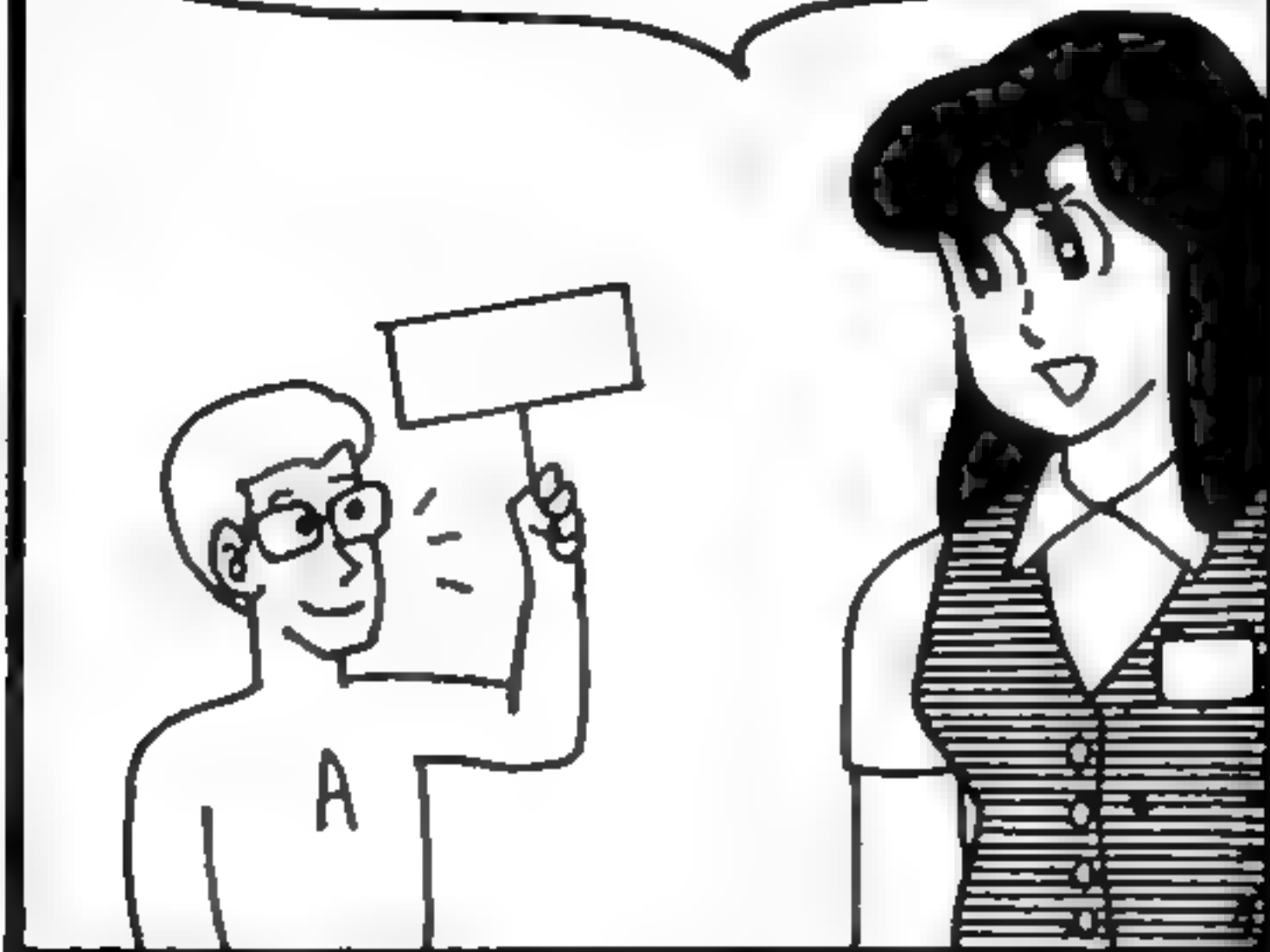
1



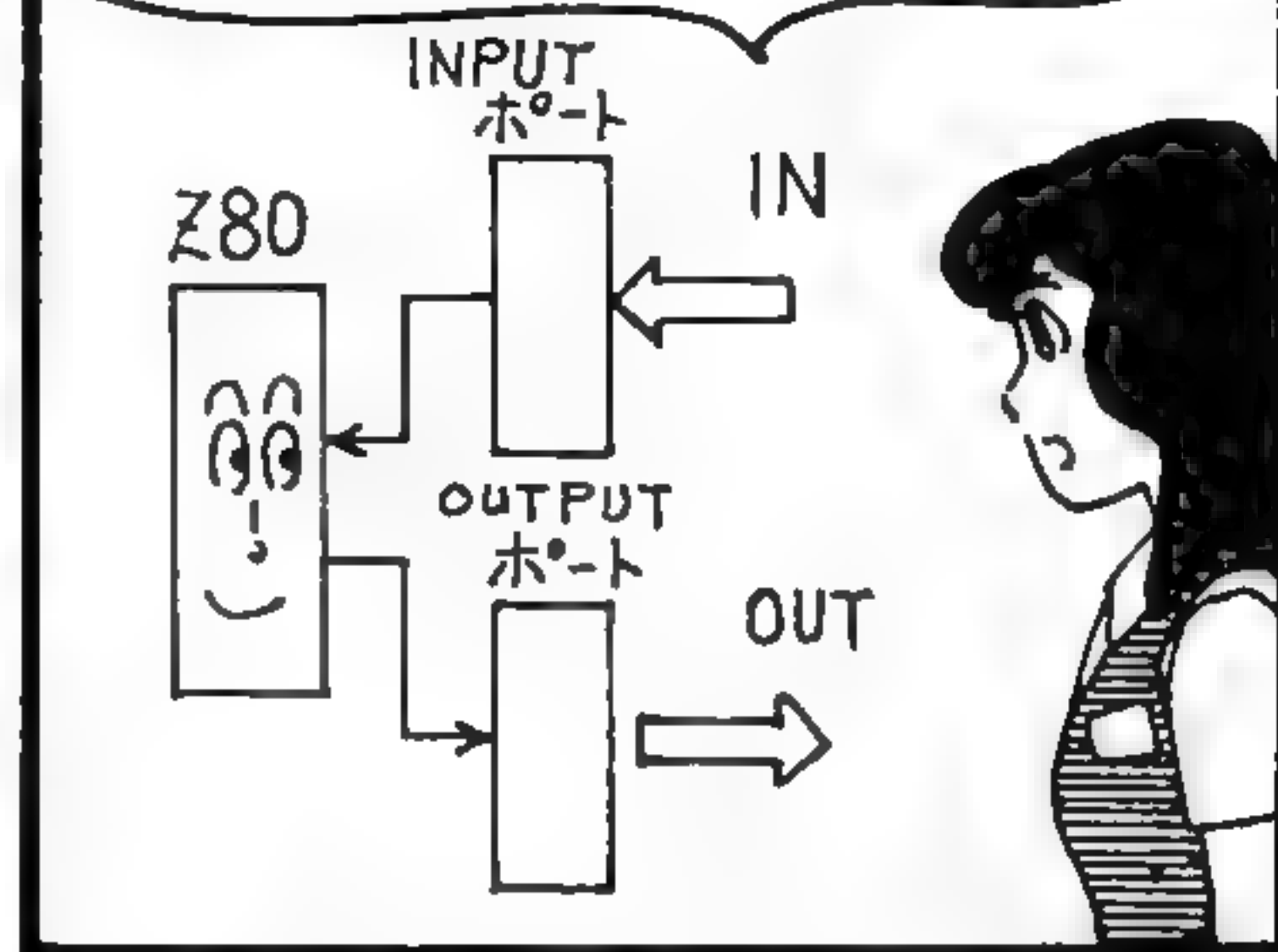
Z80では、さらにB、C、D、E、H、Lの各レジスタを介したI/O命令が追加されています。



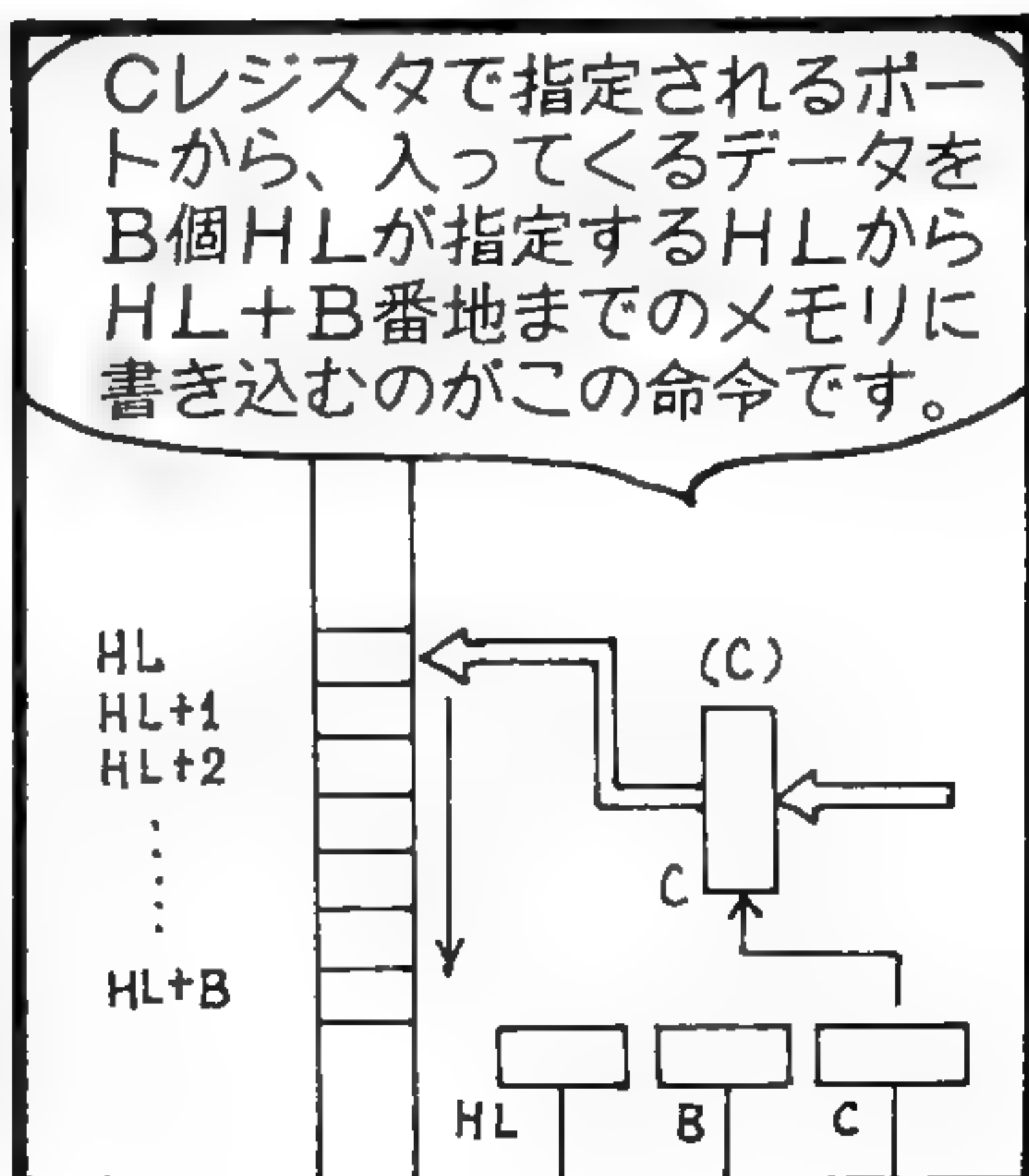
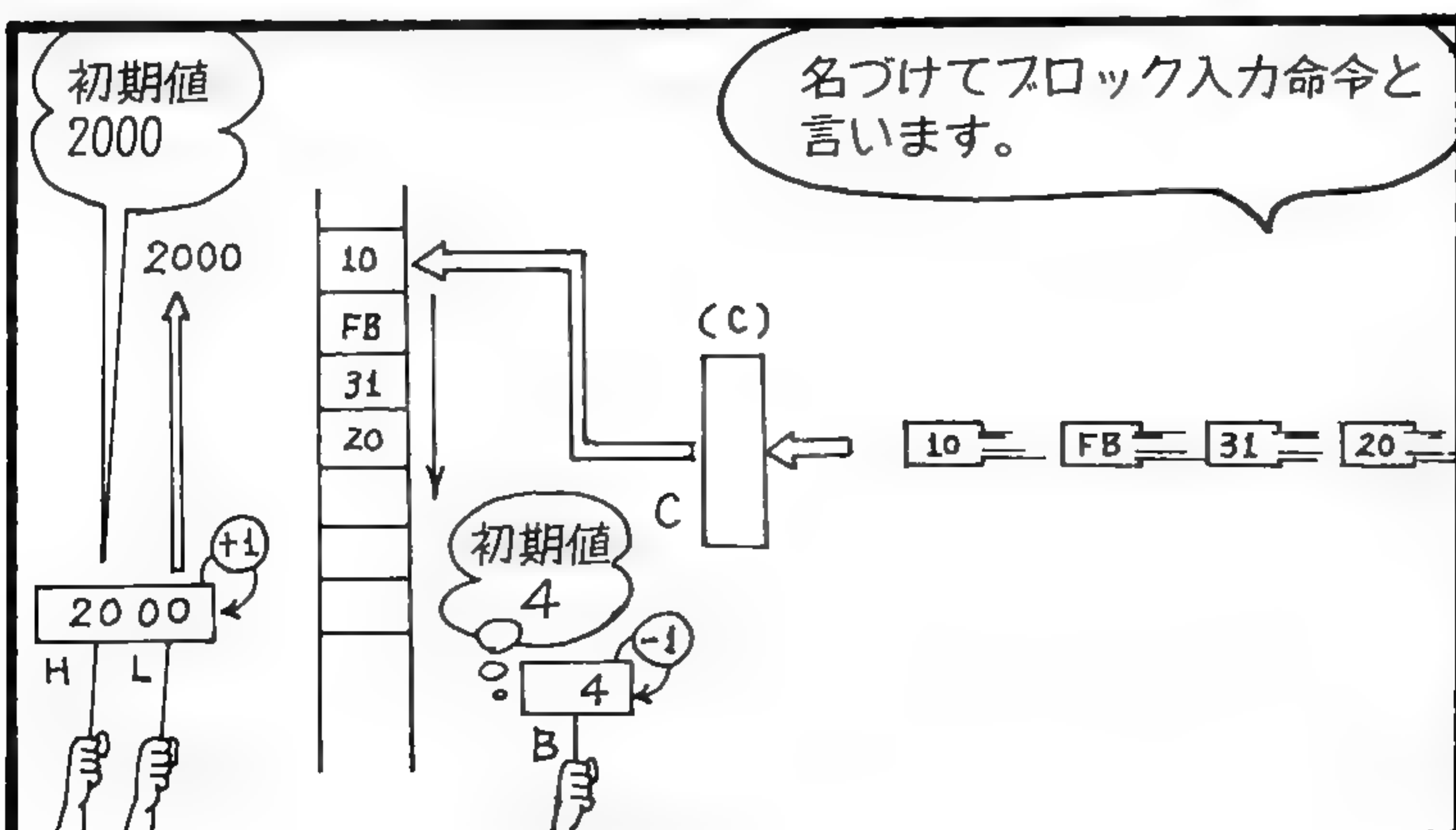
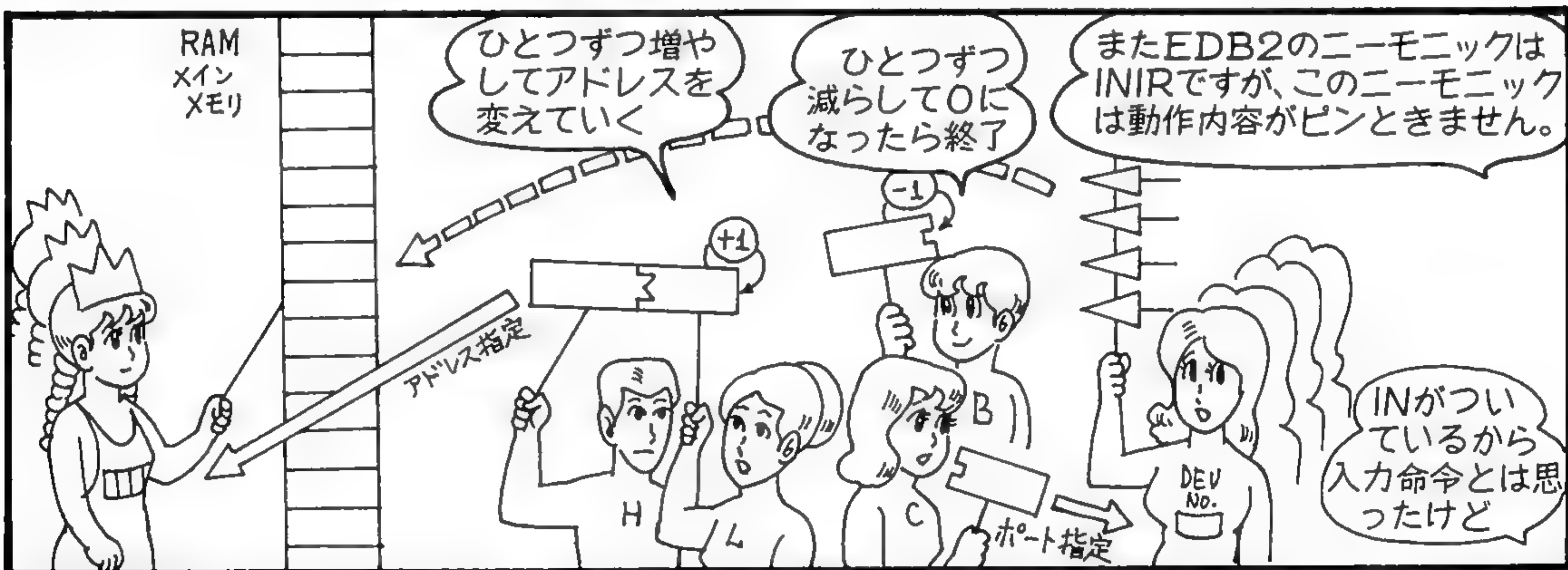
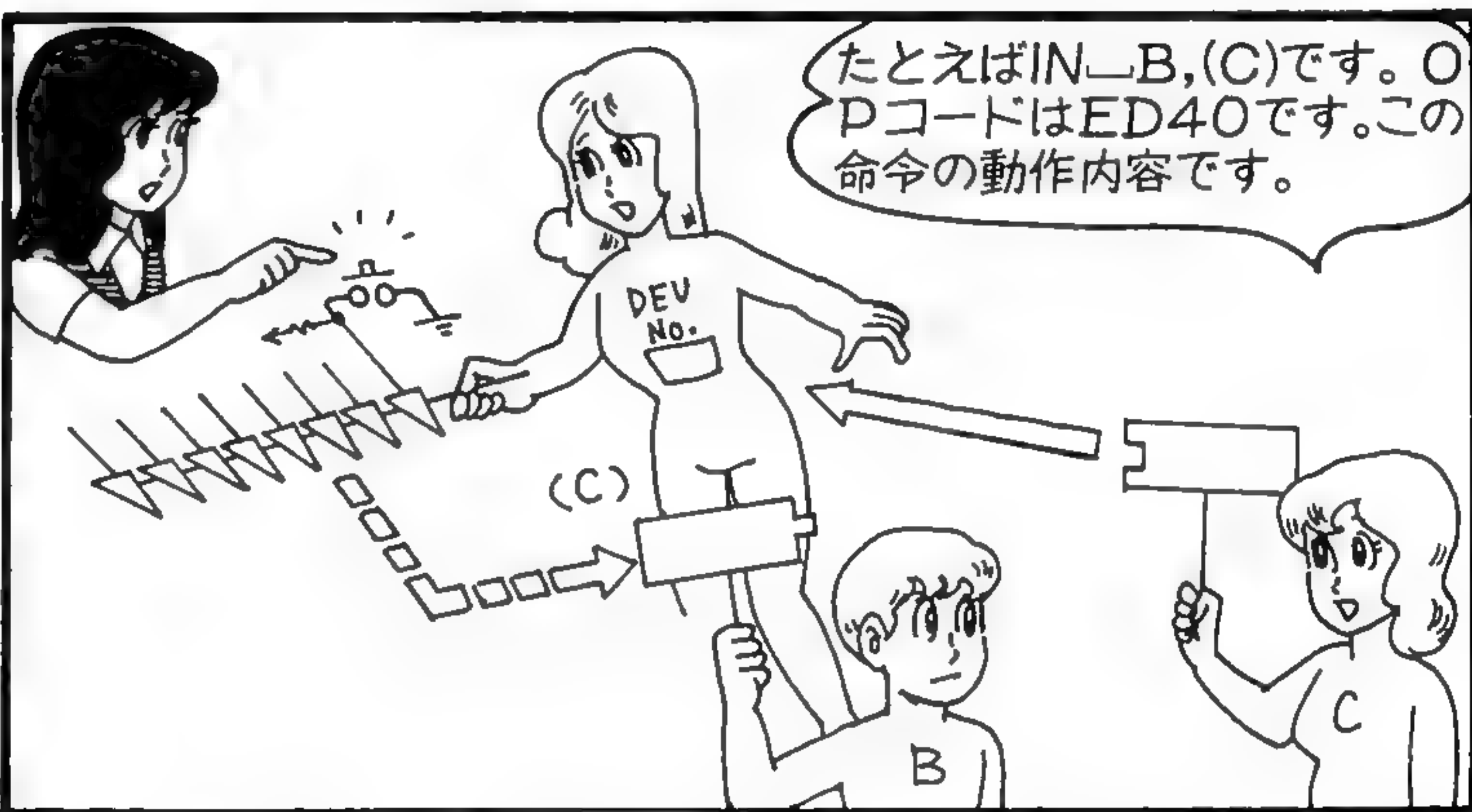
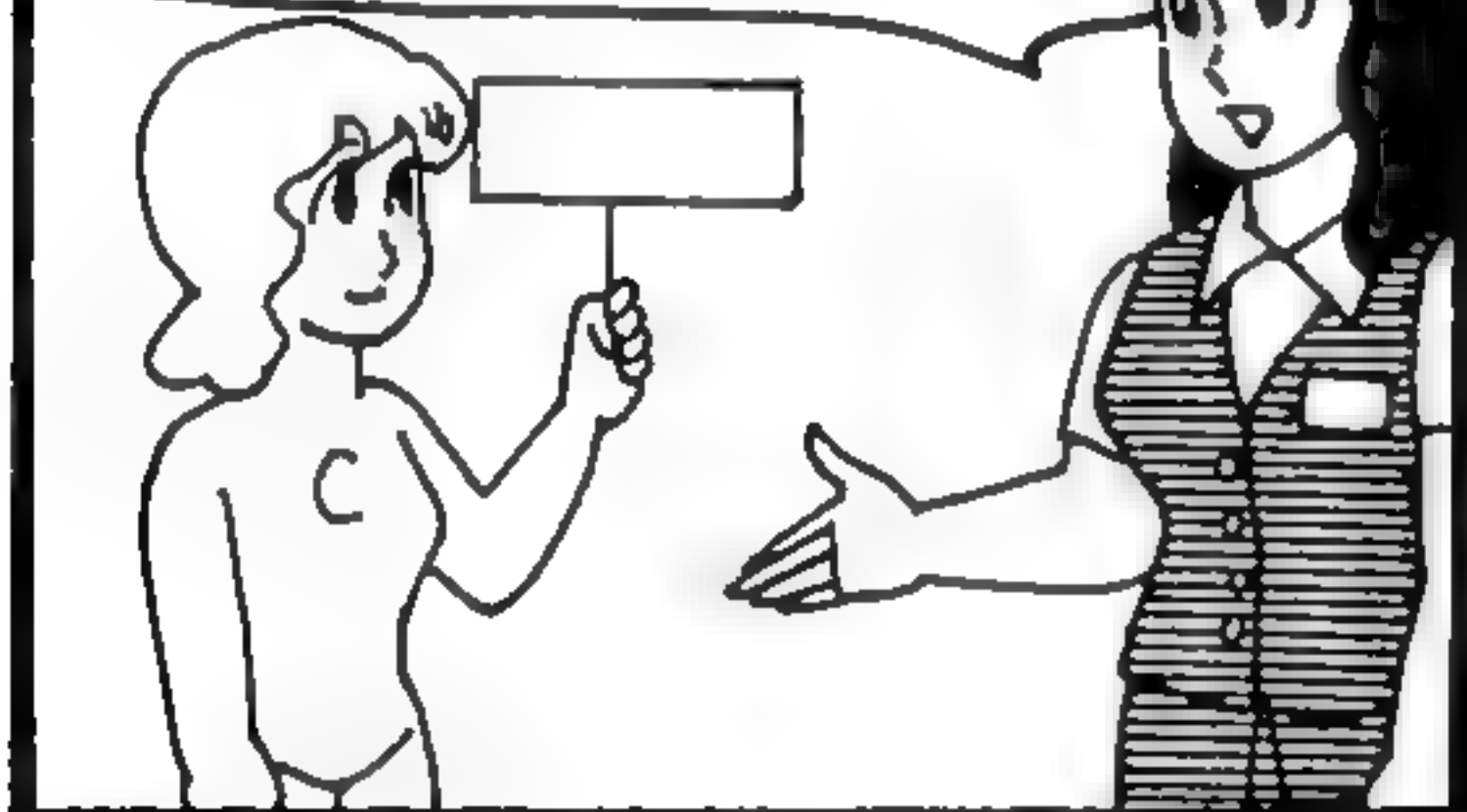
8080ではI/O命令は、D3のOUT—D8とDBのIN—D8の2つのみでした。データはどちらもAレジスタを介して出入りしました。



あとDD、ED、FDがありますがまずI/O命令に関係する、EDを1バイト目に持つOPコードを見てみます。



このED…で表現されるI/O命令の特徴はデバイスNo.の指定がCレジスタで間接的にできるということです。間接的ということは、プログラムの立場では非常に重要です。





DD 70 05 ... LD (IX+5), B

⑤

(IX) ⇒ 0  
1  
2  
3

IX+d

2013

2018をアドレス指定

FF

B

IX

コードの割当表を見てもらえばわかると思いますが、アドレス指定はIX+d、IY+dのようになっています。

残ったのは、DDとFDですが、これらは、それぞれインデックスレジスタIXとIYに関する命令です。

+d

+d

IX

IY

DDとFDのCBコードは3バイトOPコードへの入口になっていることがわかりました。内容はIX+dとIY+dのビット操作命令でした。

さらに、ですね。

DD

FD

d

FD CB d [ ]

[ ] こそ opコードが決定

8080ではHLレジスタが、このインデックスレジスタの役目をしていますが+dのような間接でかつ相対的な扱いはできませんでした。

(IX) ⇒

+d

d=0, 1, 2, ...  
...FE, FF

ブロック転送

1バイト8080コンパチCPコード 244個

248個

39個

55個

39個

結局8080とコンパチなコードは244個、Z80専用コードは451個ありました。

CB

DD

ED

FD

DD-CB-d

31個

31個

Z80専用 8個

次の6ページはZ80のコード表です。8080の命令と対比できるようにしました。

LD A, A

Aのマシン語は7F

たとえば、こんな具合です。

8ビットコード

第1オペランド 第2オペランド

		A	B	C	D	E	H	L
LD	A	7F	78	79	7A	7B	7C	7D
	B	47	40	41	42	43	44	45
	C	4F	48	49	4A	4B	4C	4D
	D							

普通Z80のテキストではこんなコードマップでなく、機能別のニモニック優先の表になっています。

Z80

Z80 マイコン実入門

Z80 アセンブラプログラミング

マイクロコンピュータZ80



8080

Z80コード表 (8080コンパチ)

このページのコード表は8080のコード表と同じです。変な言い方ですがたとえば61というコードはこの表のニーモニックではLD<sub>HL</sub>, Cですが8080コード表ではMOV<sub>HL</sub>, Cです。Z80も8080もこの61というコードに対して(H)←(C)の動作をします。つまり8080で作ったマシン語プログラムでZ80 CPUは動作するのです。

	0.	1.	2.	3.	4.	5.	6.	7.	8.	9.	A.	B.	C.	D.	E.	F.
• 0	NOF	DJNZ <sub>BC</sub>	JR <sub>NZ</sub> ,e	JR <sub>NC</sub> ,C	LD <sub>HL</sub> ,B	LD <sub>HL</sub> ,B	LD <sub>HL</sub> ,B	LD <sub>HL</sub> ,B	ADD <sub>HL</sub> ,B	SUB <sub>HL</sub> ,B	AND <sub>HL</sub> ,B	OR <sub>HL</sub> ,B	RET <sub>NZ</sub>	RET <sub>NC</sub>	RET <sub>PO</sub>	RET <sub>PL</sub>
• 1	LD <sub>BC</sub> ,n1n2	LD <sub>DE</sub> ,n1n2	LD <sub>HL</sub> ,n1n2	LD <sub>SP</sub> ,n1n2	LD <sub>HL</sub> ,C	LD <sub>HL</sub> ,C	LD <sub>HL</sub> ,C	LD <sub>HL</sub> ,C	ADD <sub>HL</sub> ,C	SUB <sub>HL</sub> ,C	AND <sub>HL</sub> ,C	OR <sub>HL</sub> ,C	POP <sub>BC</sub>	POP <sub>DE</sub>	POP <sub>HL</sub>	POP <sub>AF</sub>
• 2	LD <sub>(BC)</sub> ,A	LD <sub>(DE)</sub> ,A	LD <sub>(n1n2)</sub> ,HL	LD <sub>(n1n2)</sub> ,A	LD <sub>HL</sub> ,D	LD <sub>HL</sub> ,D	LD <sub>HL</sub> ,D	LD <sub>HL</sub> ,D	ADD <sub>HL</sub> ,D	SUB <sub>HL</sub> ,D	AND <sub>HL</sub> ,D	OR <sub>HL</sub> ,D	JP <sub>NZ</sub> ,n1n2	JP <sub>NC</sub> ,n1n2	JP <sub>PO</sub> ,n1n2	JP <sub>PL</sub> ,n1n2
• 3	INC <sub>BC</sub>	INC <sub>DE</sub>	INC <sub>HL</sub>	INC <sub>SP</sub>	LD <sub>HL</sub> ,E	LD <sub>HL</sub> ,E	LD <sub>HL</sub> ,E	LD <sub>HL</sub> ,E	ADD <sub>HL</sub> ,E	SUB <sub>HL</sub> ,E	AND <sub>HL</sub> ,E	OR <sub>HL</sub> ,E	JP <sub>NZ</sub> ,n1n2	OUT <sub>(n)</sub> ,A	EX <sub>(SP)</sub> ,HL	DI
• 4	INC <sub>B</sub>	INC <sub>D</sub>	INC <sub>H</sub>	INC <sub>(HL)</sub>	LD <sub>HL</sub> ,B	LD <sub>HL</sub> ,B	LD <sub>HL</sub> ,B	LD <sub>HL</sub> ,B	ADD <sub>HL</sub> ,B	SUB <sub>HL</sub> ,B	AND <sub>HL</sub> ,B	OR <sub>HL</sub> ,B	CALL <sub>NZ</sub> ,n1n2	CALL <sub>NC</sub> ,n1n2	CALL <sub>PO</sub> ,n1n2	CALL <sub>PL</sub> ,n1n2
• 5	DEC <sub>B</sub>	DEC <sub>D</sub>	DEC <sub>H</sub>	DEC <sub>(HL)</sub>	LD <sub>HL</sub> ,L	LD <sub>HL</sub> ,L	LD <sub>HL</sub> ,L	LD <sub>HL</sub> ,L	ADD <sub>HL</sub> ,L	SUB <sub>HL</sub> ,L	AND <sub>HL</sub> ,L	OR <sub>HL</sub> ,L	PUSH <sub>BC</sub>	PUSH <sub>DE</sub>	PUSH <sub>HL</sub>	PUSH <sub>AF</sub>
• 6	LD <sub>B</sub> ,n	LD <sub>D</sub> ,n	LD <sub>H</sub> ,n	LD <sub>(HL)</sub> ,n	LD <sub>HL</sub> ,HALT	LD <sub>HL</sub> ,HALT	LD <sub>HL</sub> ,HALT	LD <sub>HL</sub> ,HALT	ADD <sub>HL</sub> ,A	SUB <sub>HL</sub> ,A	AND <sub>HL</sub> ,A	OR <sub>HL</sub> ,A	ADD <sub>HL</sub> ,n	SUB <sub>HL</sub> ,n	AND <sub>HL</sub> ,n	OR <sub>HL</sub> ,n
• 7	RACA	RLA	DAA	SCF	LD <sub>HL</sub> ,A	LD <sub>HL</sub> ,A	LD <sub>HL</sub> ,A	LD <sub>HL</sub> ,A	ADD <sub>HL</sub> ,A	SUB <sub>HL</sub> ,A	AND <sub>HL</sub> ,A	OR <sub>HL</sub> ,A	RST0	RST2	RST4	RST6
• 8	EXAF,AF	JR <sub>OE</sub>	JR <sub>OZ</sub> ,e	JR <sub>OC</sub> ,e	LD <sub>HL</sub> ,B	LD <sub>HL</sub> ,B	LD <sub>HL</sub> ,B	LD <sub>HL</sub> ,B	ADD <sub>HL</sub> ,B	SUB <sub>HL</sub> ,B	AND <sub>HL</sub> ,B	OR <sub>HL</sub> ,B	RET <sub>NZ</sub>	RET <sub>NC</sub>	RET <sub>PE</sub>	RET <sub>PM</sub>
• 9	ADD <sub>HL</sub> ,BC	ADD <sub>HL</sub> ,DE	ADD <sub>HL</sub> ,HL	ADD <sub>HL</sub> ,SP	LD <sub>HL</sub> ,C	LD <sub>HL</sub> ,C	LD <sub>HL</sub> ,C	LD <sub>HL</sub> ,C	ADD <sub>HL</sub> ,C	SBC <sub>HL</sub> ,C	XOR <sub>HL</sub> ,C	CP <sub>HL</sub> ,C	RET	EXX	JP <sub>(HL)</sub>	LD <sub>SP</sub> ,HL
• A	LD <sub>(BC)</sub> ,A	LD <sub>A</sub> ,(DE)	LD <sub>HL</sub> ,(n1n2)	LD <sub>(n1n2)</sub> ,A	LD <sub>HL</sub> ,D	LD <sub>HL</sub> ,D	LD <sub>HL</sub> ,D	LD <sub>HL</sub> ,D	ADD <sub>HL</sub> ,D	SBC <sub>HL</sub> ,D	XOR <sub>HL</sub> ,D	CP <sub>HL</sub> ,D	JP <sub>NZ</sub> ,n1n2	JP <sub>NC</sub> ,n1n2	JP <sub>PE</sub> ,n1n2	JP <sub>PL</sub> ,n1n2
• B	DEC <sub>BC</sub>	DEC <sub>DE</sub>	DEC <sub>HL</sub>	DEC <sub>SP</sub>	LD <sub>HL</sub> ,E	LD <sub>HL</sub> ,E	LD <sub>HL</sub> ,E	LD <sub>HL</sub> ,E	ADD <sub>HL</sub> ,E	SBC <sub>HL</sub> ,E	XOR <sub>HL</sub> ,E	CP <sub>HL</sub> ,E	CBA <sub>0-3</sub>	IN <sub>A</sub> ,(n)	EX <sub>HL</sub>	EI
• C	INC <sub>C</sub>	INC <sub>E</sub>	INC <sub>L</sub>	INC <sub>A</sub>	LD <sub>HL</sub> ,H	LD <sub>HL</sub> ,H	LD <sub>HL</sub> ,H	LD <sub>HL</sub> ,H	ADD <sub>HL</sub> ,H	SBC <sub>HL</sub> ,H	XOR <sub>HL</sub> ,H	CP <sub>HL</sub> ,H	CALL <sub>NZ</sub> ,n1n2	CALL <sub>NC</sub> ,n1n2	CALL <sub>PE</sub> ,n1n2	CALL <sub>PL</sub> ,n1n2
• D	DEC <sub>C</sub>	DEC <sub>E</sub>	DEC <sub>L</sub>	DEC <sub>A</sub>	LD <sub>HL</sub> ,L	LD <sub>HL</sub> ,L	LD <sub>HL</sub> ,L	LD <sub>HL</sub> ,L	ADD <sub>HL</sub> ,L	SBC <sub>HL</sub> ,L	XOR <sub>HL</sub> ,L	CP <sub>HL</sub> ,L	CALL <sub>NZ</sub> ,n1n2	CALL <sub>NC</sub> ,n1n2	CALL <sub>PE</sub> ,n1n2	CALL <sub>PL</sub> ,n1n2
• E	LD <sub>C</sub> ,n	LD <sub>E</sub> ,n	LD <sub>L</sub> ,n	LD <sub>A</sub> ,n	LD <sub>HL</sub> ,C	LD <sub>HL</sub> ,C	LD <sub>HL</sub> ,C	LD <sub>HL</sub> ,C	ADD <sub>HL</sub> ,C	SBC <sub>HL</sub> ,C	XOR <sub>HL</sub> ,C	CP <sub>HL</sub> ,C	ADC <sub>HL</sub> ,n	SBC <sub>HL</sub> ,n	XOR <sub>HL</sub> ,n	CP <sub>HL</sub> ,n
• F	RACA	RRA	CPL	CCF	LD <sub>HL</sub> ,A	LD <sub>HL</sub> ,A	LD <sub>HL</sub> ,A	LD <sub>HL</sub> ,A	ADD <sub>HL</sub> ,A	SBC <sub>HL</sub> ,A	XOR <sub>HL</sub> ,A	CP <sub>HL</sub> ,A	RST1	RST3	RST5	RST7

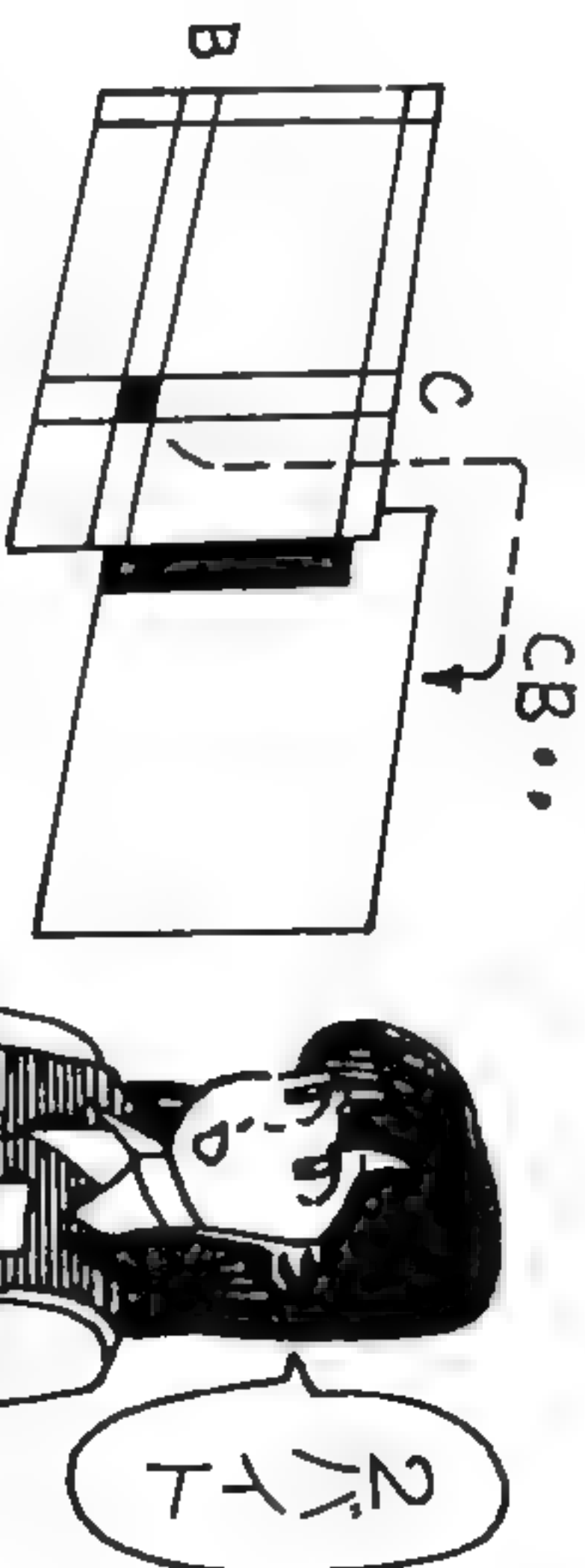


# Z80 CPUのニーモニック-OPコード割当表(Z80専用コード)

## CBペーシ

この表にあるニーモニックのOPコードはすべて2バイトです。1バイト目がCB 2バイト目がこの表のコードです。

たとえば、BIT 6,BはCB 70というOPコードになります。



	0.	1.	2.	3.	4.	5.	6.	7.	8.	9.	A.	B.	C.	D.	E.	F.
• 0	RLC B	RL B	SLA B		BIT 0,B	BIT 2,B	BIT 4,B	BIT 6,B	RES 0,B	RES 2,B	RES 4,B	RES 6,B	SET 0,B	SET 2,B	SET 4,B	SET 6,B
• 1	RLC C	RL C	SLA C		BIT 0,C	BIT 2,C	BIT 4,C	BIT 6,C	RES 0,C	RES 2,C	RES 4,C	RES 6,C	SET 0,C	SET 2,C	SET 4,C	SET 6,C
• 2	RLC D	RL D	SLA D		BIT 0,D	BIT 2,D	BIT 4,D	BIT 6,D	RES 0,D	RES 2,D	RES 4,D	RES 6,D	SET 0,D	SET 2,D	SET 4,D	SET 6,D
• 3	RLC E	RL E	SLA E		BIT 0,E	BIT 2,E	BIT 4,E	BIT 6,E	RES 0,E	RES 2,E	RES 4,E	RES 6,E	SET 0,E	SET 2,E	SET 4,E	SET 6,E
• 4	RLC H	RL H	SLA H		BIT 0,H	BIT 2,H	BIT 4,H	BIT 6,H	RES 0,H	RES 2,H	RES 4,H	RES 6,H	SET 0,H	SET 2,H	SET 4,H	SET 6,H
• 5	RLC L	RL L	SLA L		BIT 0,L	BIT 2,L	BIT 4,L	BIT 6,L	RES 0,L	RES 2,L	RES 4,L	RES 6,L	SET 0,L	SET 2,L	SET 4,L	SET 6,L
• 6	RLC (HL)	RL (HL)	SLA (HL)		BIT 0,(HL)	BIT 2,(HL)	BIT 4,(HL)	BIT 6,(HL)	RES 0,(HL)	RES 2,(HL)	RES 4,(HL)	RES 6,(HL)	SET 0,(HL)	SET 2,(HL)	SET 4,(HL)	SET 6,(HL)
• 7	RLC A	RL A	SLA A		BIT 0,A	BIT 2,A	BIT 4,A	BIT 6,A	RES 0,A	RES 2,A	RES 4,A	RES 6,A	SET 0,A	SET 2,A	SET 4,A	SET 6,A
• 8	RRC B	RR B	SRA B	SRL B	BIT 1,B	BIT 3,B	BIT 5,B	BIT 7,B	RES 1,B	RES 3,B	RES 5,B	RES 7,B	SET 1,B	SET 3,B	SET 5,B	SET 7,B
• 9	RRC C	RR C	SRA C	SRL C	BIT 1,C	BIT 3,C	BIT 5,C	BIT 7,C	RES 1,C	RES 3,C	RES 5,C	RES 7,C	SET 1,C	SET 3,C	SET 5,C	SET 7,C
• A	RRC D	RR D	SRA D	SRL D	BIT 1,D	BIT 3,D	BIT 5,D	BIT 7,D	RES 1,D	RES 3,D	RES 5,D	RES 7,D	SET 1,D	SET 3,D	SET 5,D	SET 7,D
• B	RRC E	RR E	SRA E	SRL E	BIT 1,E	BIT 3,E	BIT 5,E	BIT 7,E	RES 1,E	RES 3,E	RES 5,E	RES 7,E	SET 1,E	SET 3,E	SET 5,E	SET 7,E
• C	RRC H	RR H	SRA H	SRL H	BIT 1,H	BIT 3,H	BIT 5,H	BIT 7,H	RES 1,H	RES 3,H	RES 5,H	RES 7,H	SET 1,H	SET 3,H	SET 5,H	SET 7,H
• D	RRC L	RR L	SRA L	SRL L	BIT 1,L	BIT 3,L	BIT 5,L	BIT 7,L	RES 1,L	RES 3,L	RES 5,L	RES 7,L	SET 1,L	SET 3,L	SET 5,L	SET 7,L
• E	RRC (HL)	RR (HL)	SRA (HL)	SRL (HL)	BIT 1,(HL)	BIT 3,(HL)	BIT 5,(HL)	BIT 7,(HL)	RES 1,(HL)	RES 3,(HL)	RES 5,(HL)	RES 7,(HL)	SET 1,(HL)	SET 3,(HL)	SET 5,(HL)	SET 7,(HL)
• F	RRC A	RR A	SRA A	SRL A	BIT 1,A	BIT 3,A	BIT 5,A	BIT 7,A	RES 1,A	RES 3,A	RES 5,A	RES 7,A	SET 1,A	SET 3,A	SET 5,A	SET 7,A

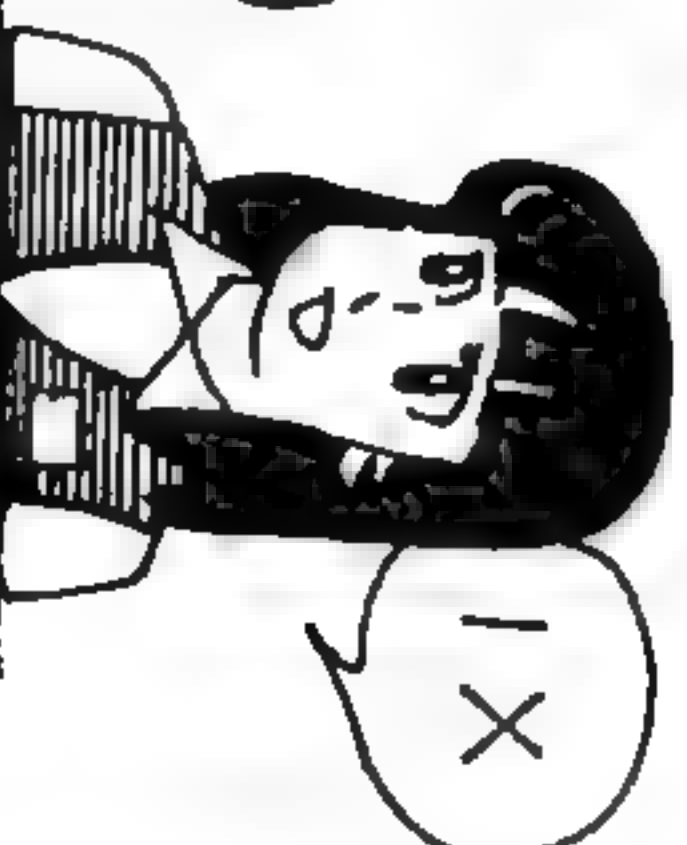


DDV-1

このページはインテックスレジス  
CBコードはさらに次のページ  
への入力です。



LDIX, n1n2の  
OPコ-トは  
DD21n2n1でず。

[illegible]



# Z80 CPUのニーモニック-OPコード割当表

## EDページ (Z80専用コード)



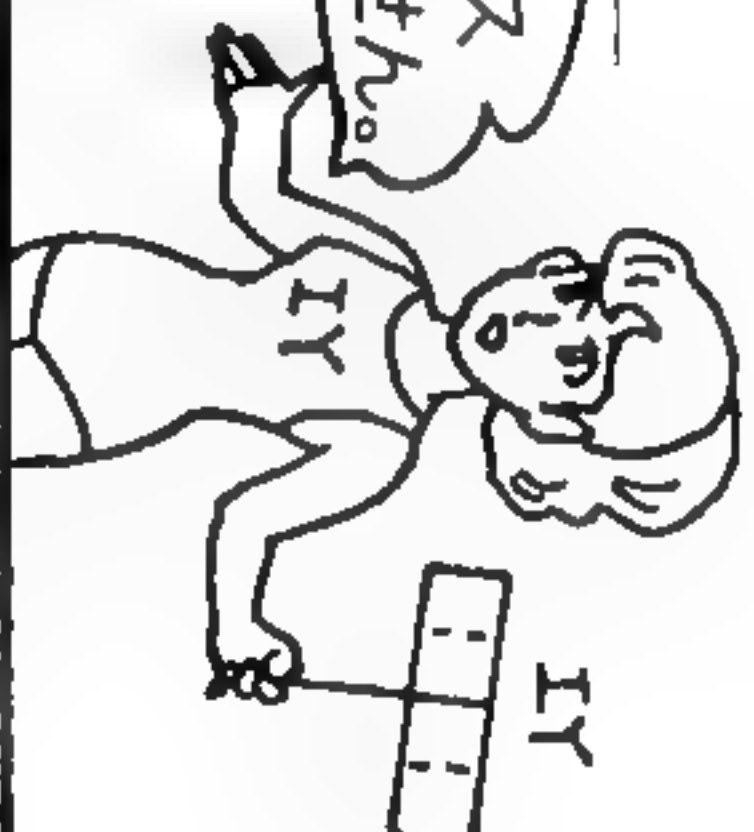
	0.	1.	2.	3.	4.	5.	6.	7.	8.	9.	A.	B.	C.	D.	E.	F.
.0					IN <sub>u</sub> B,(C)	IN <sub>u</sub> D,(C)	IN <sub>u</sub> H,(C)				LDI	LDIR				
.1					OUT <sub>u</sub> (C),B	OUT <sub>u</sub> (C),D	OUT <sub>u</sub> (C),H				CPI	CPDR				
.2					SBC <sub>u</sub> HL,BC	SBC <sub>u</sub> HL,DE	SBC <sub>u</sub> HL,HL	SBC <sub>u</sub> HL,SP			INI	INIR				
.3					LD <sub>u</sub> (min2),BC	LD <sub>u</sub> (min2),DE		LD <sub>u</sub> (min2),SP			OUTI	OTIR			EX <sub>u</sub> (SP),IX	
.4					NEG											
.5					IM0	IM1										
.6					LD <sub>u</sub> I,A	LD <sub>u</sub> A,I	R <sub>u</sub> RD <sub>u</sub> (HL)									
.7					IN <sub>u</sub> C,(C)	IN <sub>u</sub> E,(C)	IN <sub>u</sub> L,(C)	IN <sub>u</sub> A,(C)			LDD	LDDR				
.8					OUT <sub>u</sub> (C),C	OUT <sub>u</sub> (C),E	OUT <sub>u</sub> (C),L	OUT <sub>u</sub> (C),A			CPD	CPDR				
.9					ADC <sub>u</sub> HL,BC	ADC <sub>u</sub> HL,DE	ADC <sub>u</sub> HL,HL	ADC <sub>u</sub> HL,SP			IND	INDR				
.A					LD <sub>u</sub> BC,(min2)	LD <sub>u</sub> DE,(min2)		LD <sub>u</sub> SP,(min2)			OUTD	OTDR				
.B																
.C																
.D																
.E					IM2											
.F					LD <sub>u</sub> R,A	LD <sub>u</sub> A,R	R <sub>u</sub> LD <sub>u</sub> (HL)									



## FD-302a

このFDページはインテックスレジスタ  
Yの操作命令です。

IVはアドレスではありませんが、CBコードはさらに次のページへの入口です。


$$\frac{IY+d}{IY+d}$$
[illegible]



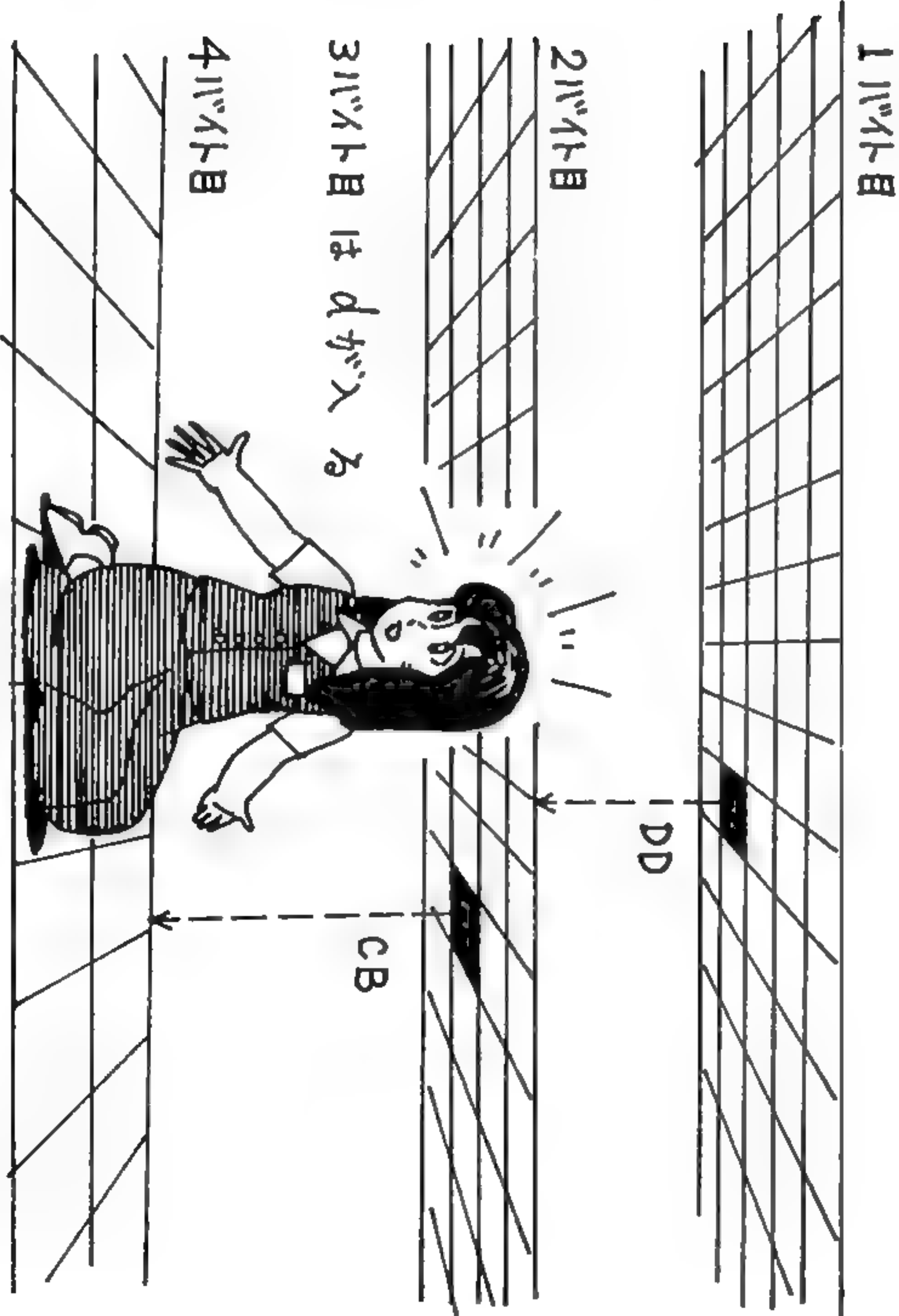
Z80 CPUのニーモニック-OPコード割当表(Z80専用コード)

DD CB d ページ

	0.	1.	2.	3.	4.	5.	6.	7.	8.	9.	A.	B.	C.	D.	E.	F.
	RLC <sub>u</sub> (IX+d)	RL <sub>u</sub> (IX+d)	SLA <sub>u</sub> (IX+d)	SRL <sub>u</sub> (IX+d)	BIT <sub>u</sub> 0,(IX+d)	BIT <sub>u</sub> 2,(IX+d)	BIT <sub>u</sub> 4,(IX+d)	BIT <sub>u</sub> 6,(IX+d)	RES <sub>u</sub> 0,(IX+d)	RES <sub>u</sub> 2,(IX+d)	RES <sub>u</sub> 4,(IX+d)	RES <sub>u</sub> 6,(IX+d)	SET <sub>u</sub> 0,(IX+d)	SET <sub>u</sub> 2,(IX+d)	SET <sub>u</sub> 4,(IX+d)	SET <sub>u</sub> 6,(IX+d)
・6					BIT <sub>u</sub> 1,(IX+d)	BIT <sub>u</sub> 3,(IX+d)	BIT <sub>u</sub> 5,(IX+d)	BIT <sub>u</sub> 7,(IX+d)	RES <sub>u</sub> 1,(IX+d)	RES <sub>u</sub> 3,(IX+d)	RES <sub>u</sub> 5,(IX+d)	RES <sub>u</sub> 7,(IX+d)	SET <sub>u</sub> 1,(IX+d)	SET <sub>u</sub> 3,(IX+d)	SET <sub>u</sub> 5,(IX+d)	SET <sub>u</sub> 7,(IX+d)
・E	RRC <sub>u</sub> (IX+d)	RR <sub>u</sub> (IX+d)	SRA <sub>u</sub> (IX+d)													

この2つのページのニーモニックの命令は3バイトのOPコードになります。

たとえばニーモニック SET<sub>u</sub>4,(IX+d) のOPコードは DD CB d E6 になります。



FD CB d ページ

	0.	1.	2.	3.	4.	5.	6.	7.	8.	9.	A.	B.	C.	D.	E.	F.
	RLC <sub>u</sub> (IX+d)	RL <sub>u</sub> (IX+d)	SLA <sub>u</sub> (IX+d)	SRL <sub>u</sub> (IX+d)	BIT <sub>u</sub> 0,(IX+d)	BIT <sub>u</sub> 2,(IX+d)	BIT <sub>u</sub> 4,(IX+d)	BIT <sub>u</sub> 6,(IX+d)	RES <sub>u</sub> 0,(IX+d)	RES <sub>u</sub> 2,(IX+d)	RES <sub>u</sub> 4,(IX+d)	RES <sub>u</sub> 6,(IX+d)	SET <sub>u</sub> 0,(IX+d)	SET <sub>u</sub> 2,(IX+d)	SET <sub>u</sub> 4,(IX+d)	SET <sub>u</sub> 6,(IX+d)
・6					BIT <sub>u</sub> 1,(IX+d)	BIT <sub>u</sub> 3,(IX+d)	BIT <sub>u</sub> 5,(IX+d)	BIT <sub>u</sub> 7,(IX+d)	RES <sub>u</sub> 1,(IX+d)	RES <sub>u</sub> 3,(IX+d)	RES <sub>u</sub> 5,(IX+d)	RES <sub>u</sub> 7,(IX+d)	SET <sub>u</sub> 1,(IX+d)	SET <sub>u</sub> 3,(IX+d)	SET <sub>u</sub> 5,(IX+d)	SET <sub>u</sub> 7,(IX+d)
・E	RRC <sub>u</sub> (IX+d)	RR <sub>u</sub> (IX+d)	SRA <sub>u</sub> (IX+d)													



## ⑫ 6502と6809

	A	8ビット
	X	//
	Y	//
PC(H)	PC(L)	16ビット
01	SP	8ビット
	F/F	//

6502 CPU内部レジスタ

命令数は全部で151個と8080より少ないですね。



6502はモステクノロジー社の製品でワンボードブームの最中にパソコン用に作られたCPUです。



apple



6809 CPU内部レジスタ

X		16ビット
Y		//
SP(U)		//
SP(S)		//
PC		//
A	B	8ビット×2
	DP	8ビット
	F/F	//

6809はZ80と双璧を成しているCPUです。



6809はモトローラ社のCPUです。日本では富士通や日立のパソコンなどに採用されています。

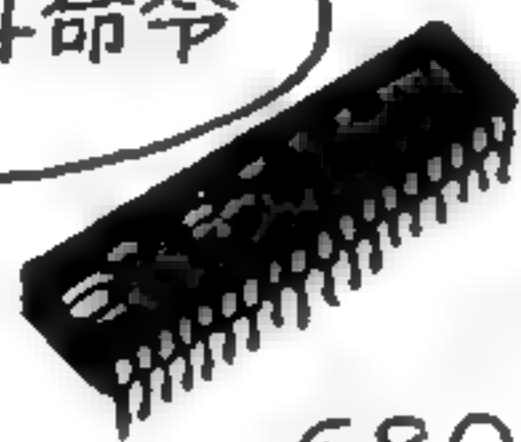


まず、レジスタの数がZ80にくらべると少ないですね。たったこれだけのレジスタで8ビットの究極と言われる命令群を持っているのです。そこでZ80の考え方の延長では理解できない部分がでてきます。



そこで6809についても少しだけその特徴的な所をお話してみたいと思います。

1464命令



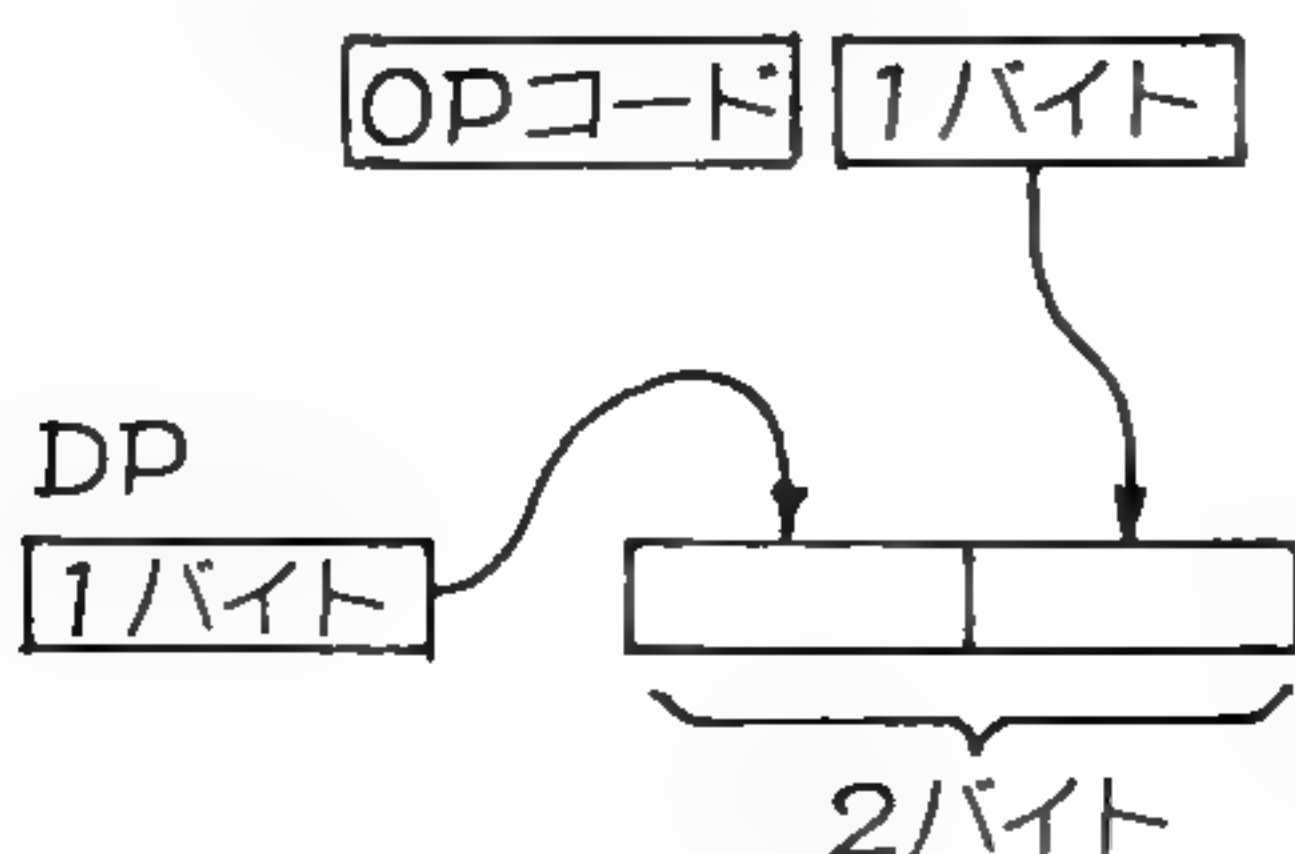
6809

基本ソフトウェアの充実からZ80が優位を保っていますが、やがて6809の基本ソフトウェアが充実してくればそのシェアは変わっていくでしょう。

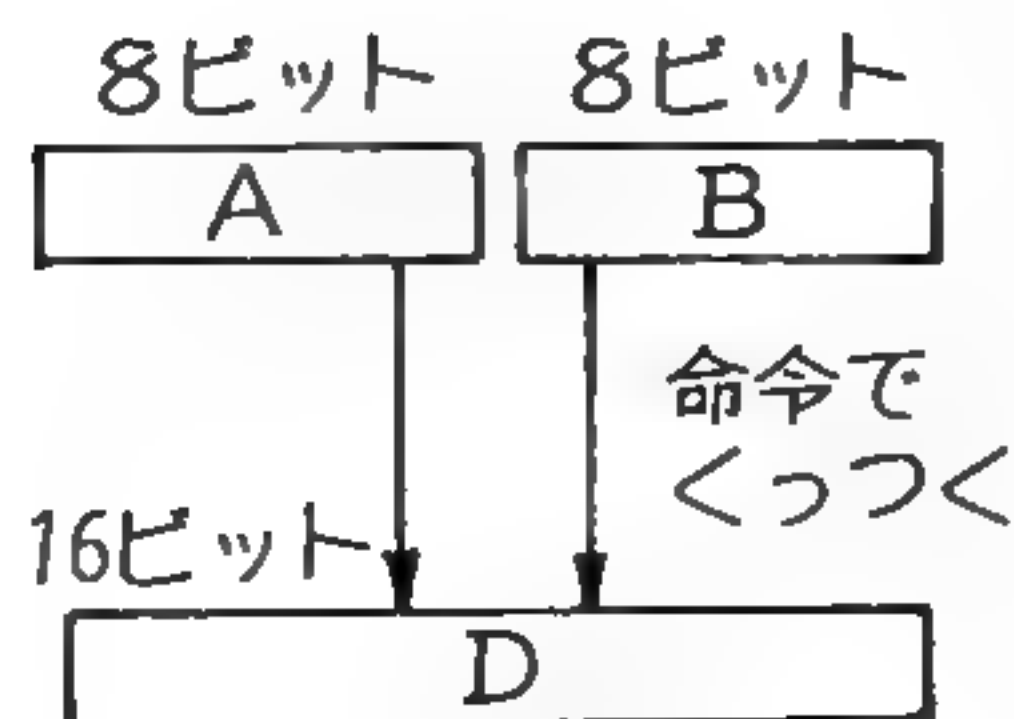


DPはダイレクトページレジスタでアドレスの上桁をこれで指定するようにしたものです。

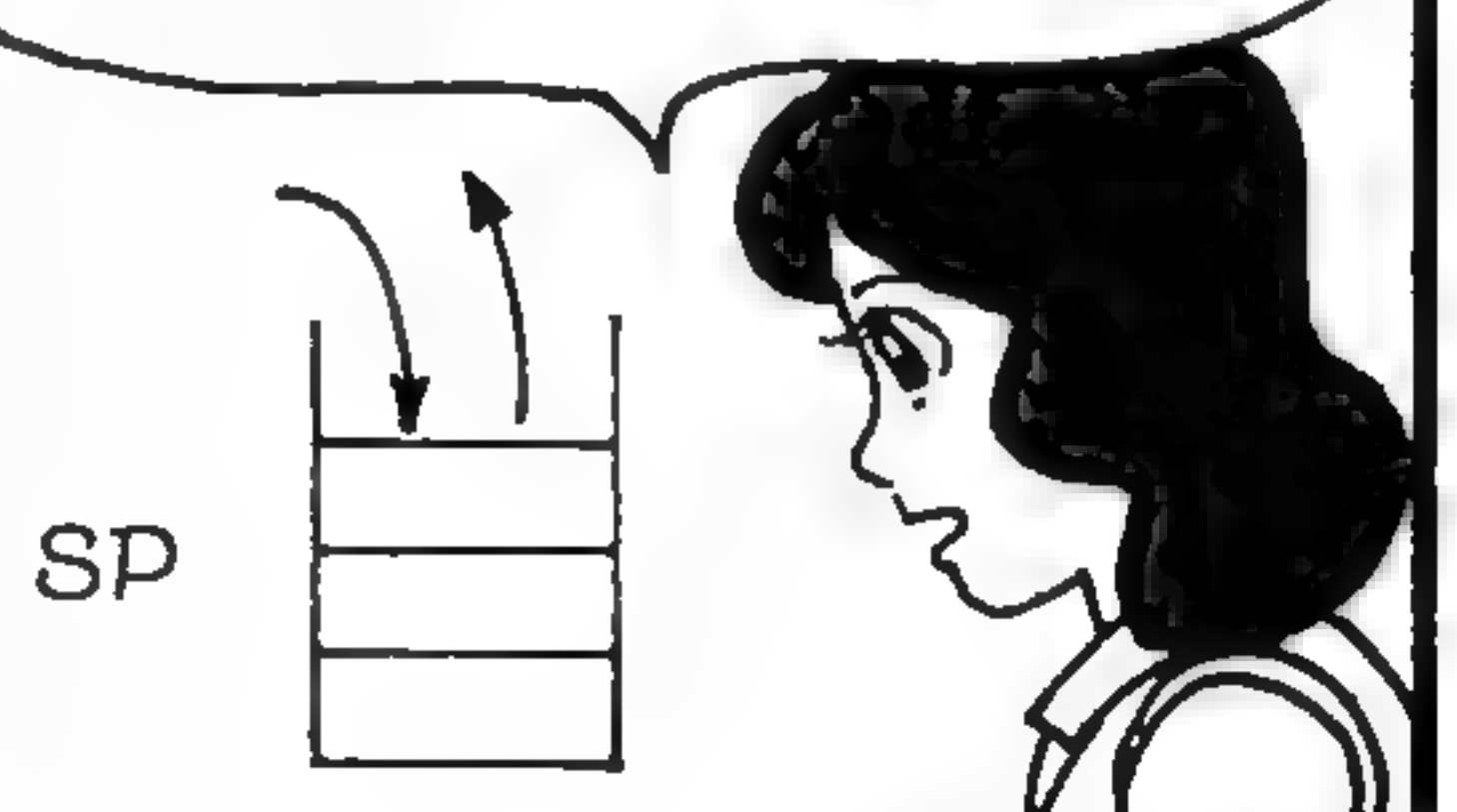
プログラム



次に変わっているのは、アキュムレータがAとBの2つありこれがくっつくとDという16ビットのアキュムレータになるということです。



まずスタックポインタが2つあり、ひとつはシステム用、ひとつはユーザー用となっています。スタックポインタの役目はZ80と変わらないのでこれ以上お話しません。





インハラントは1バイト命令です。レジスタ操作命令です。

ニーモニック OPコード  
MUL 3D

動作内容



でも、このうちのいくつかはZ80と同じはずですから中身がわかれば新しく覚えなければならないものはひとつかふたつです。

初心者にとって一番イヤラシイのがアドレッシングモードの種類がたくさんあることです。

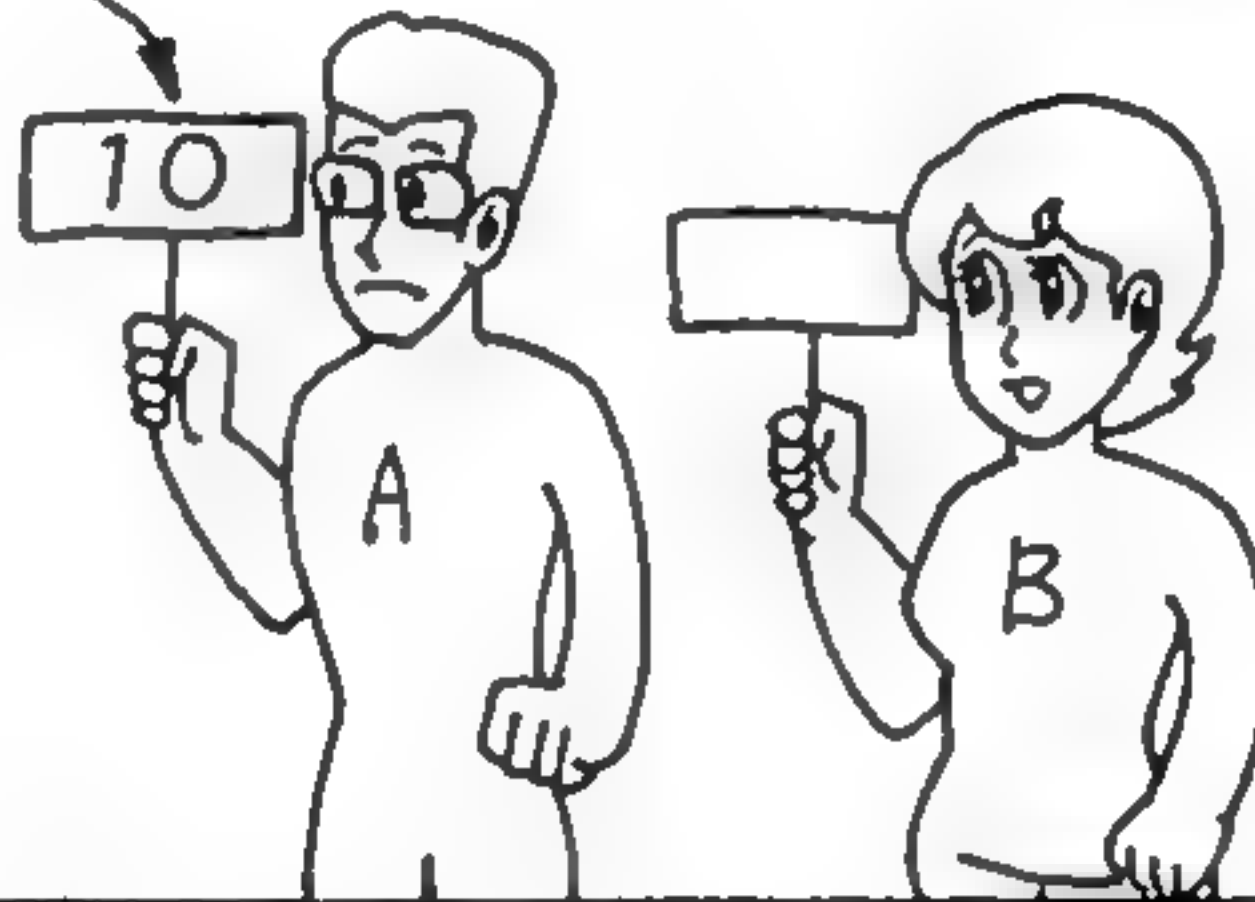
- ・インハラント
- ・インデックス
- ・イミディエイト
- ・インデックス
- ・ダイレクト
- ・インダイレクト
- ・エクステンド
- ・エクステンド
- ・インデックス
- ・インダイレクト



ニーモニック	OPコード	動作内容
LDA #10	86 10	M→A 定数をAに入れる。

この#記号がイミディエイトモードであることを示す。したがってOPコードは86となる。

イミディエイトはオペランドに直接数値を書く命令です。ニーモニックでは#記号をつけ他のモードと区別します。

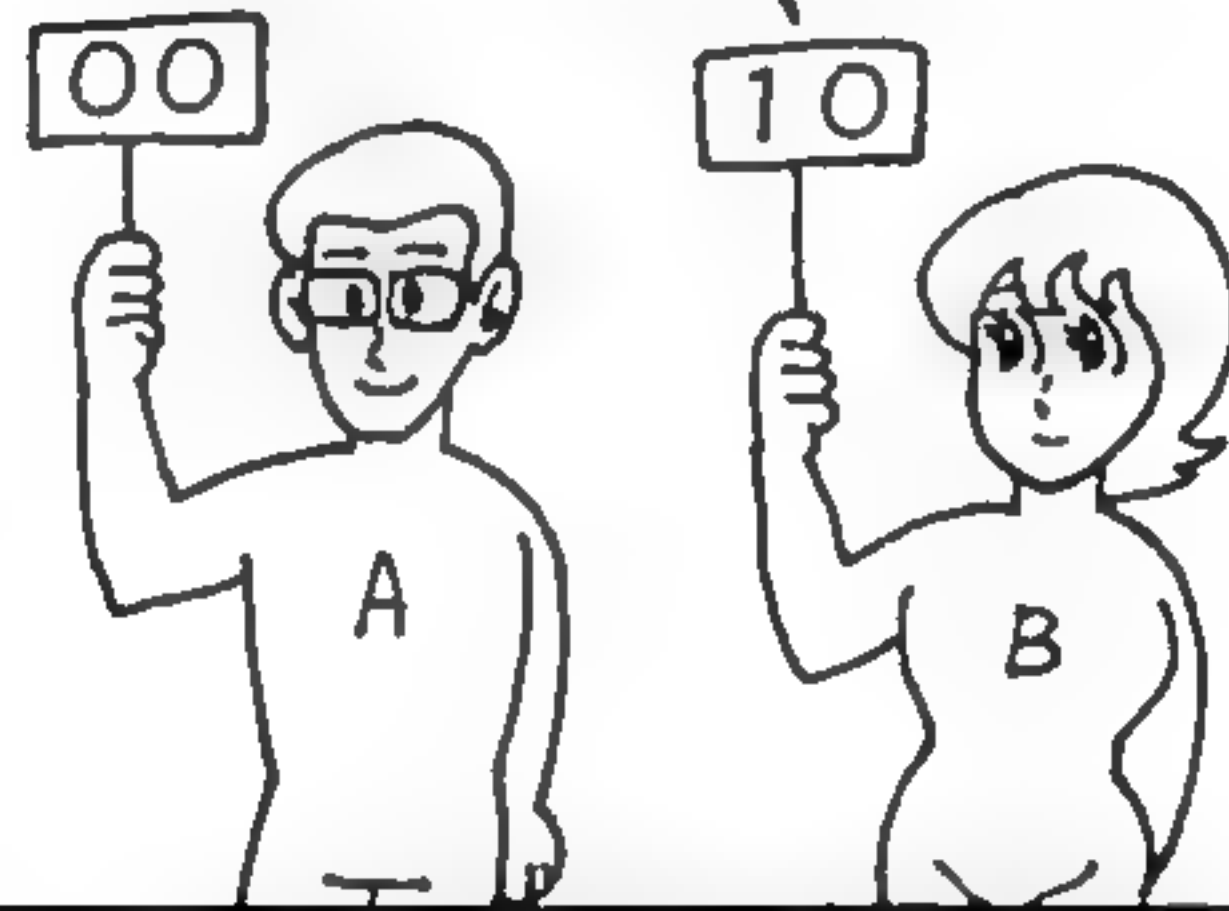


次のエクステンドは間接アドレス指定です。>記号で区別されます。

ニーモニック	OPコード	動作内容
LDA >10	B6 00 10	M→A



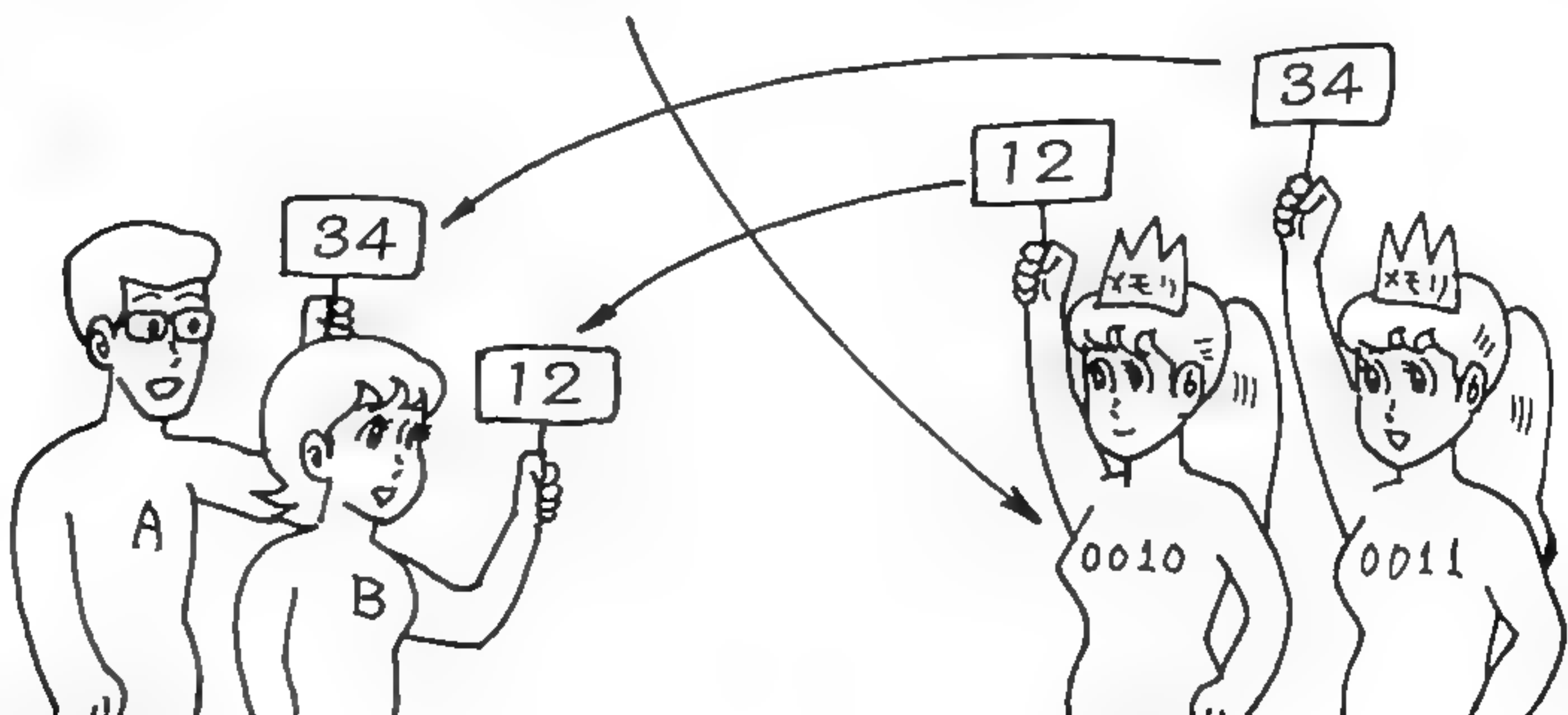
ニーモニック	OPコード	内容
LDD #0010	CC 00 10	M→D



Z80の時はニーモニックをマシン語に直すとアドレスの上桁下桁が入れかわりましたが6809ではそのままです。



ニーモニック	OPコード	動作内容
LDD >10	FC 00 10	M→D





次のインデックスモードが非常に難解に思えたモードです。バイト数の後に+記号がついているのです。

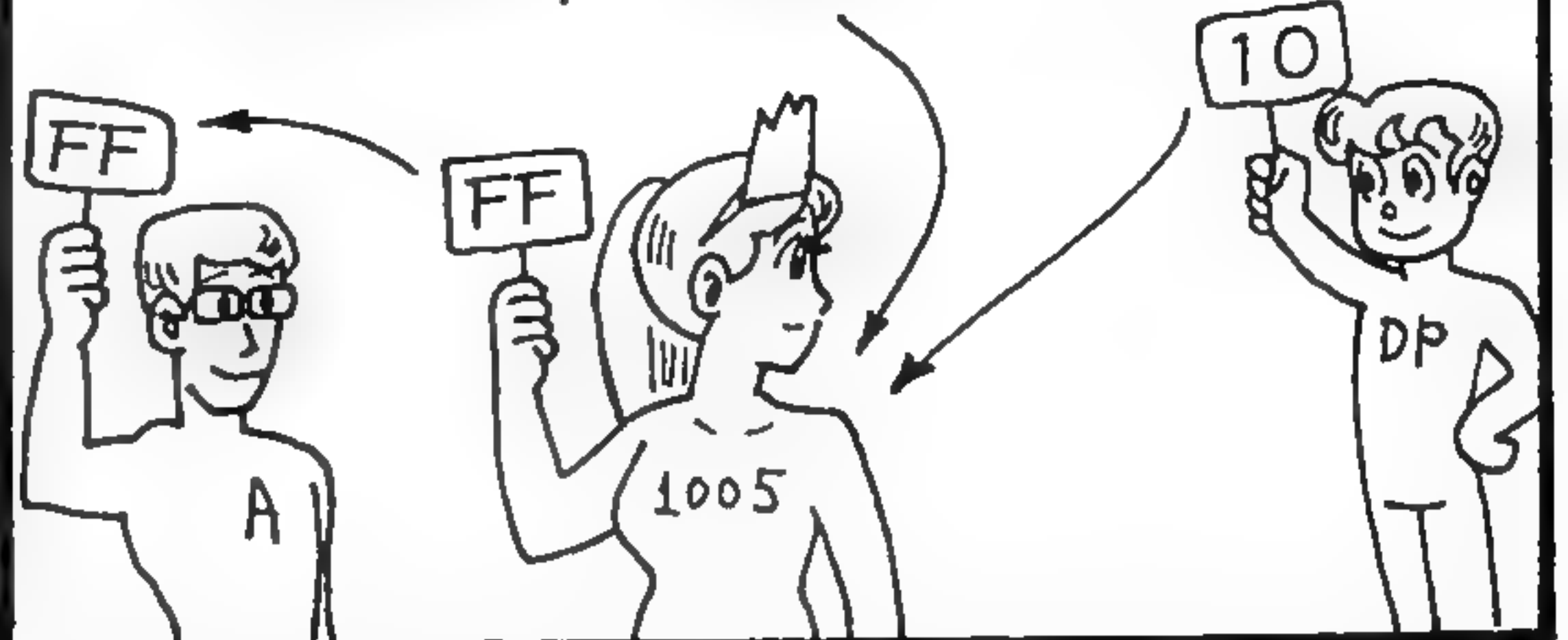
命令一覧表

インデックス		
オペコード	サイクル数	バイト数
A6	4+	2+

?

次はダイレクトモードでニーモニクの識別記号は\$です。

ニーモニク	OPコード	内容
LDA-\$05	9605	M→A



アドレス ンクモード オフ セット の種類	インデックス インダイレクト	
	ポストバイト (2バイト目の OPコード)	追加 バイト
16ビット オフセット	1rr1 1001	2

このrrはX、Y、U、Sの各レジスタです。

オペコードA6のニーモニクはLDAですがオペランドはこれに続くコード（ポストバイト）がないと決まりません。

A6+

このあとの2バイトに0120を入れるとA6990120というコードができます。ニーモニクはLDA-[\$0120,X]です。

[\$0120,X]  
↑  
16ビット  
オフセット  
インダイレクト記号

たとえばrrにXを入れると10011001となり99というポストバイトができます。

A6 99 [ ] [ ]

レジスタのコードはこうになっています。

rr	コード
X	00
Y	01
U	10
S	11

ニーモニク	OPコード	内容
LDA-[\$0120,X]	A6 99 0120	M→A

こういうのをオフセットといいます。

2013

0120  
+2013  
2133

まどろっこしいね。

28

28

3FFF

インダイレクト(間接)アドレッシング

3F

FF

2133

2134



リラティブモードといい1バイトのものがショート、2バイトのものがロングです。

ニーモニック	OPコード	内容
BEQ—30FF	1025 30FF	Z=1なら 30FF+(PC)にジャンプ



$$\begin{array}{r} 30FF \\ + 1003 \\ \hline 4102 \end{array}$$

Z=1のときの  
ジャンプ先

もうひとつジャンプ命令が特徴的です。68系ではブランチ命令と言っているものです。



またFM-11やS1で走るOS-9というUNIXライクなOSもあります。マルチタスクやマルチウィンドーの機能を持っているそうです。

ミニコンOS  
(UNIX)  
↓  
パソコンOS  
(OS-9)



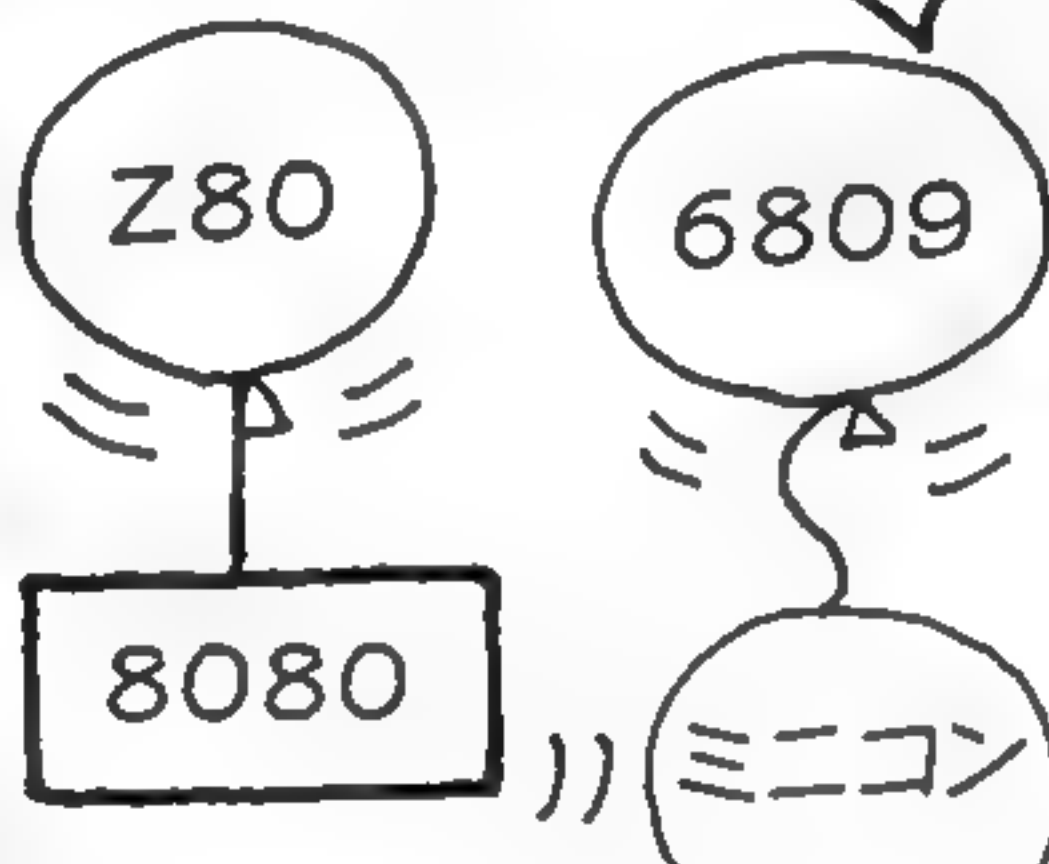
開発ツールとしてはCOMAS-FDTなどがありますが私は体験していないのでここでは触れません。

マシン語

.....



こうしてみると6809はZ80にくらべレジスタは少ないのですが、アドレッシングに大きな特徴があるようです。

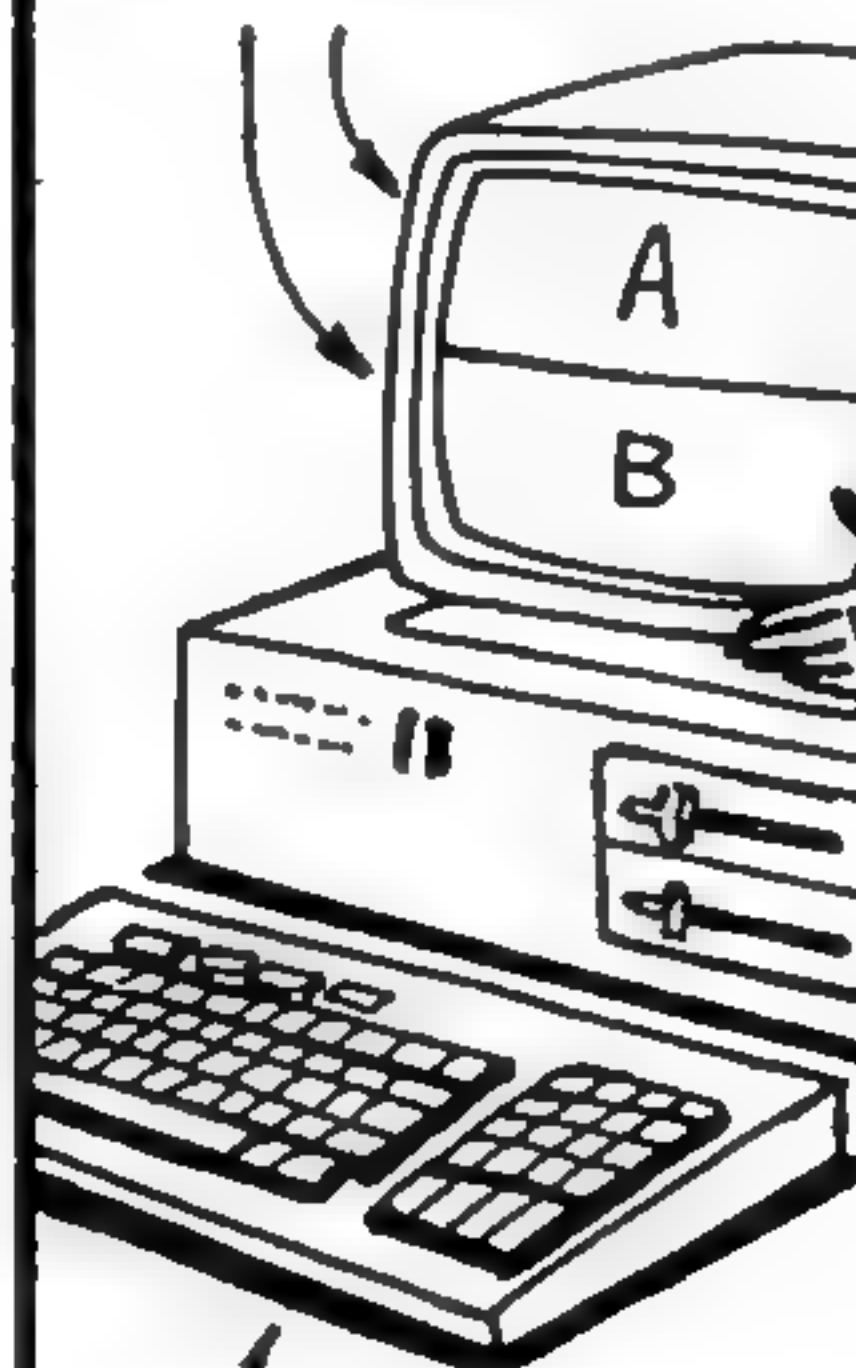


現在68系のパソコンでCP/Mを走らせるためにはCPUカードをZ80に換える必要がありますが、そのうち6809用基本ソフトが完成すれば6809の性能をいかに発揮することでしょう。



6809の命令は初めのうちはとっつきにくいかもしれませんが慣れると素晴らしいものであることがわかってくるそうです。

マルチウィンドー  
別個に表示



マルチタスク

たとえばプリンタが印字中でも別のプログラムを走らせることができる。





(6809命令一覧表の一部)

	インハラント		ダイレクト		エクステンド		イミディエイト		インデックス		リラティブ	
	OP コード	バイト 数	OP コード	バイト 数	OP コード	バイト 数	OP コード	バイト 数	OP コード	バイト 数	OP コード	バイト 数
			96	2	B6	3	86	2	A6	2+		



友子ちゃんの疑問

ニーモニックはLDA。  
オペランド部はどのよ  
うに書いたらいいのか  
しら？

オフセットっ  
て何かしら？

インデックスの  
バイト数2+の  
+は何かしら？

6809アセンブリ言語 記述例					
命 令	モード	ダイレクト	エクステンド	イミディエイト	インデックス
	オペランド 識別記号	\$	>	#	オフセット値 レジスタ [ , ] インダイレクト 記号
LDA (M→A)	ニーモニック 表記	LDA_ \$25	LDA_ >25	LDA_ #10	LDA_ [ \$0120, X ]
	マシン語 表記	96 25	B6 25	86 10	A6 99 01 20

68系のマシン語に  
変換しても80系の  
ように上桁下桁が逆  
転しない。理由は、  
CPUが上位アドレ  
スを先に読むため  
です。



## マシン語Q&A

### \* マシン語とは？

8ビットCPUでは8ビットで表現されるコードでコンピュータが理解できるON、OFFの信号を表わしたものです。たとえば0011 1110というコードをオペレーションコード読み込み（フェッチ）のタイミングでCPUが読み込むと、CPUは次の1バイト（8ビット）のデータをAレジスタに入れるという動作をします。しかし、普通は4ビットずつに区切って16進2桁で表わし、これをマシン語として使っています。0011 1110は3Eと表現します。（16進でプログラムする時はモニタ（プログラム）が必要です。）

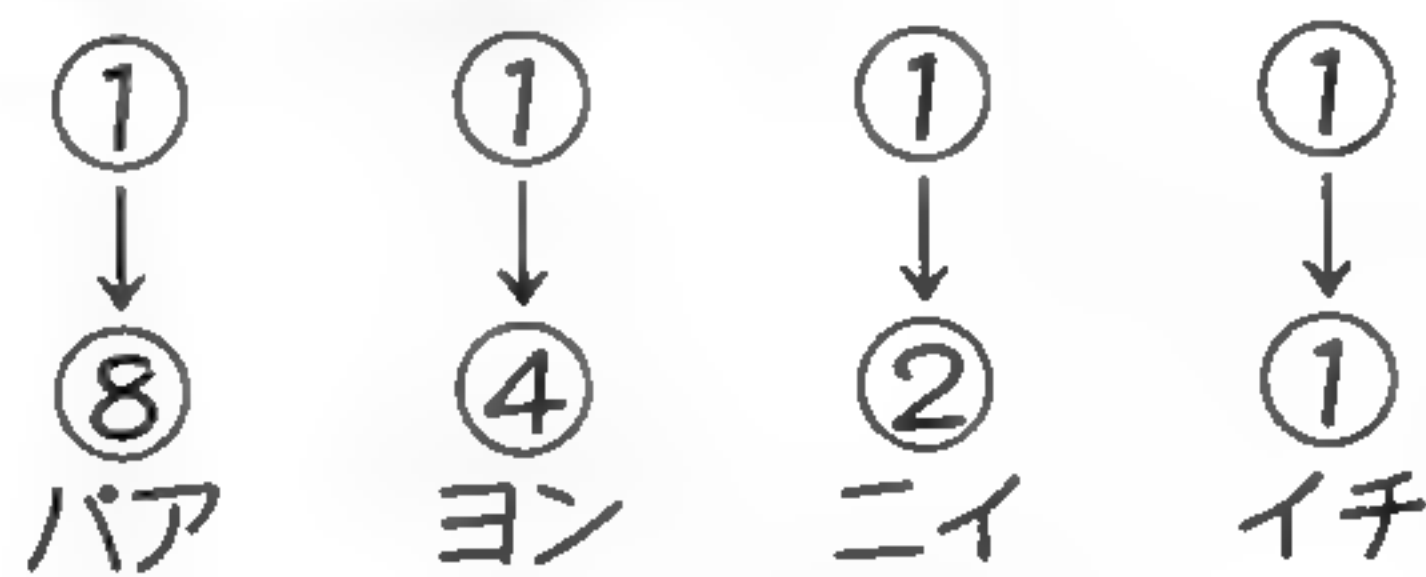
### \* 2進数と16進数の関連は

2進数は0か1で表現されますが、これを‘ゼロ’と‘イチ’と発音すると10進数に慣れている私たちの頭の中は混乱してしまいます。電圧信号と思って‘ロー’と‘ハイ’と発音するとなんとなくこれまでの数字とは違う世界が見えてくるのではないのでしょうか。

10進	2進	16進
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

$$\begin{array}{r} 0 \\ +0 \\ \hline 0 \end{array} \quad \begin{array}{r} 1 \\ +0 \\ \hline 0 \end{array} \quad \begin{array}{r} 1 \\ +1 \\ \hline 10 \end{array}$$

2進→10進換算のしかた



全部足すと15です。  
10進に換算した時の  
‘重み’

ビットの位置で  
‘重み’に差がある  
んです。

バラバラに置いたのではコードの意味がなくなってしまう。





\* ニーモニック言語とは

3Eといった16進でプログラムした方が0011 1110という2進よりはわかりやすいのですが、プログラムする時はまだよいとしてもデバックが大変です。そこで英文字でCPUの動作内容がわかるような形で表現したものが、MVI Aというニーモニックなのです。ニーモニックとコードとの間にはCPUを作った会社の変換ルールしかないため、変換表がなくては変換できません。8080、Z80、6809、それぞれニーモニックが違います。このニーモニックコードは、マシン語コードと1対1で対応しているため、これもマシン語という場合もあります。

\* どのCPUのマシン語を覚えるのがよいか

初心者はZ80のニーモニックが使いやすいのではないかと思います。Z80は参考書もたくさん出ているため勉強するにはよい環境です。MSXではCPUとしてZ80相当品を使うことになっています。6809の場合はまだ環境がととのっていないので、よけいな労力を払うことになるかもしれませんが……。

\* どうしたらマシン語を使えるか

ワンボードマイコンの時代にはいやでも16進のマシン語でCPUとトークする必要がありました。ROM型パソコンを持っている人はMONコマンドで16進のマシン語が使えます。(ハンドヘルドやポケコンにはMONモードはありません。) CP/Mの走るパソコンならニーモニック言語が使えます。実用的な使い方としては(BASIC言語 + マシン語)というパターンになります。

\* マシン語は統一されるか

家庭用電源は西日本と東日本では周波数が違いますね。誰も60Hzか50Hzのどちらかに統一しようとは思わないでしょう。影響が大きすぎます。日本では8ビットCPUはZ80を含む80(ハチマル)系と68(ロクハチ)系の2つのマシン語があります。JIS規格で検討されるようになったらあるいはどちらかに統一されるかもしれませんが、あえてその必要もないでしょう。





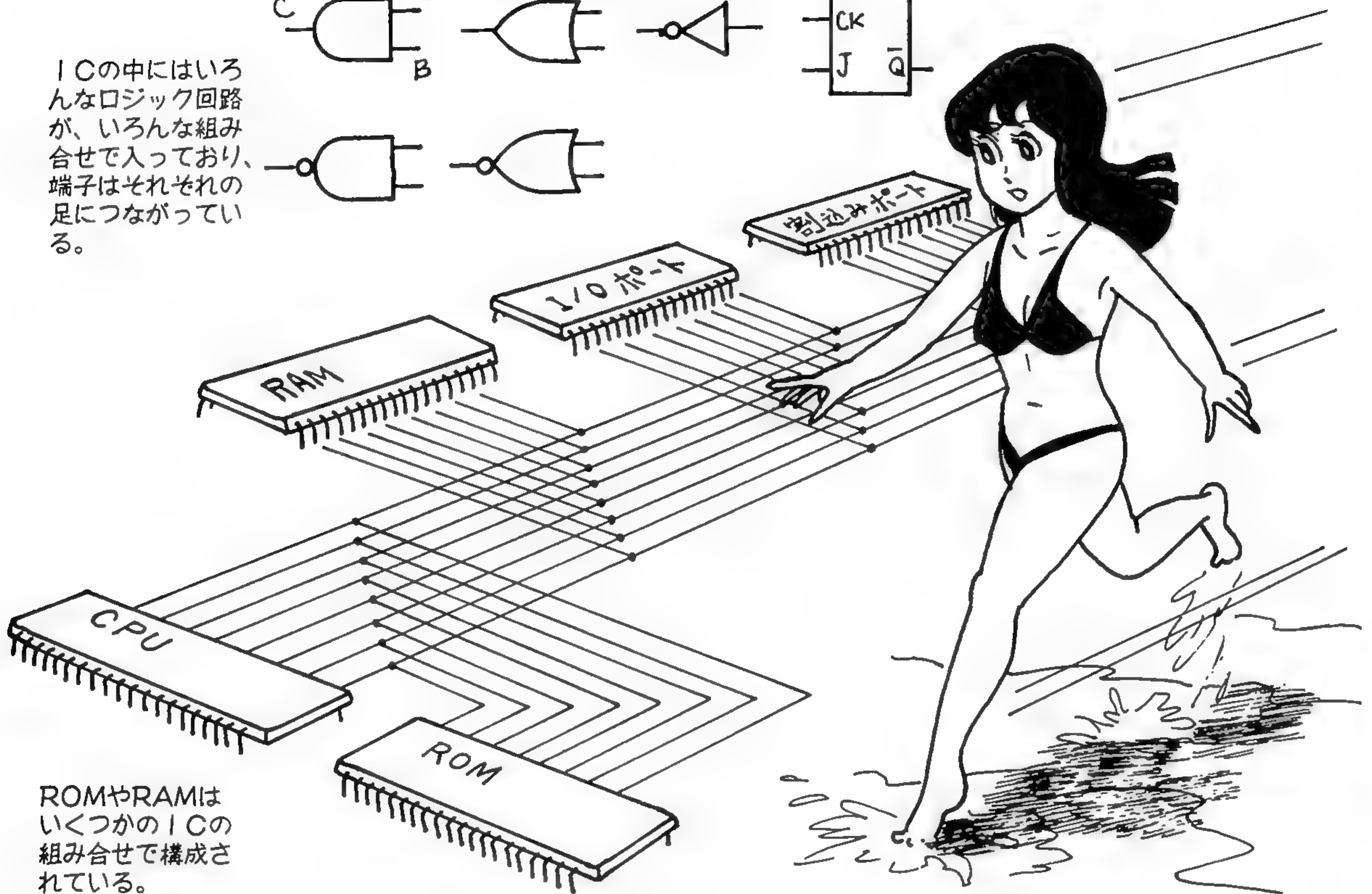
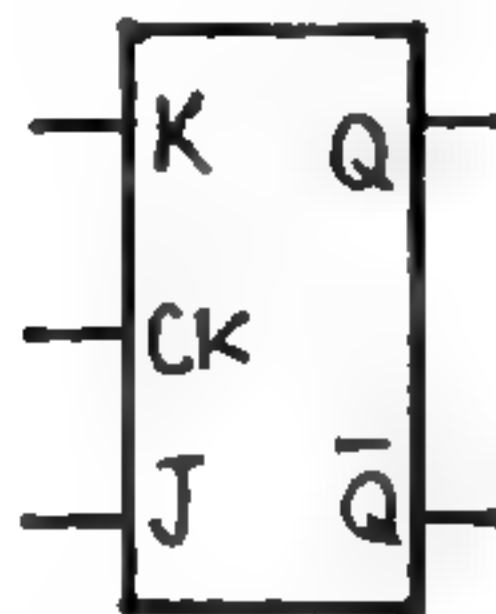
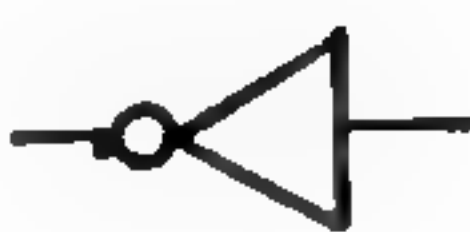
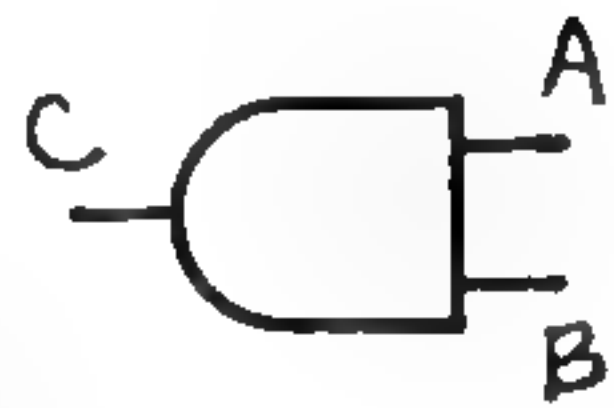
# □CPUの中はどうなっているか

ロジック

回路の種類

フリップ・フロップ

ICの中にはいろいろなロジック回路が、いろんな組み合わせで入っており、端子はそれぞれの足につながっている。



ROMやRAMはいくつかのICの組み合わせで構成されている。

回路のハードの説明は、ちょっとむずかしいので、興味のない人は適当にとばして読んでください。



SSS

Z-80	2,800.-
8085	4,000.-
8080	1,800.-
8224	400.-
8228	1,200.-

ICの技術は日進月歩もいいとこ、とても目まぐるしく改廃が行なわれています。価格も、どんどん変わっていきますよ。

## ①石(IC)の大きさ



ちょっとむずかしいのが、フリップ・フロップ。ICの集積度も比較的高くなっています。

14、16ピン

一番簡単な論理素子がゲートと呼ばれるものです。ICの集積度も少ないものです。

だから日ごろからどんなICがあるか見慣れておかなければ、必要な時にすぐに回路を組むことができません。

が、ほしい。

IC回路図に、-368、-74、-174、-LS368、-08、-04とか書いてあるのは、すべてTI社のもので前にSN74が略されているのです。

SN74368  
SN7474  
SN74174  
SN74LS368  
SN7408  
SN7404

ゲートやフリップ・フロップなんかはテキサス・インスツルメンツ社の製品番号で示するのが一般的です。たとえばANDゲートなら-00、これはSN7400のことを指しています。

テキサス インスツルメンツ社  
社の略を  
TI社

ところがマイコンのCPUクラスになると、とても、とても。ハハハ…(と、笑ってゴマかすの)

CPUのハード的なお話をする前に回路を構成するのに必要なICについての予備知識を説明しましょう。

こんな小さなパッケージの中にどんどん情報が詰め込まれていきつつあります。これに何を入れたら何が出てくるのかをしらねばなりません。

また、先頭に8のついていて、たとえば8080、8216、8224などはインテル社のものです。ICの場合、この番号が社名や仕様を表わす大事な意味を持っています。もちろん、これは登録商標にもなっています。

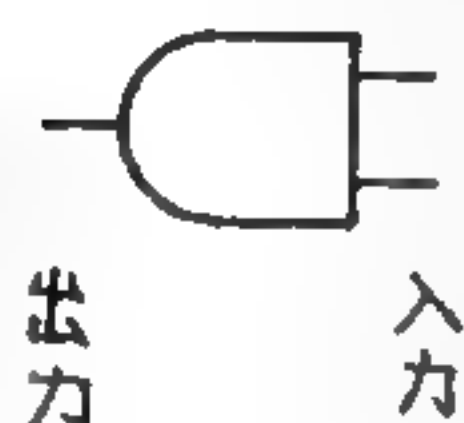
8080  
8212  
8251  
8255  
8259  
8257



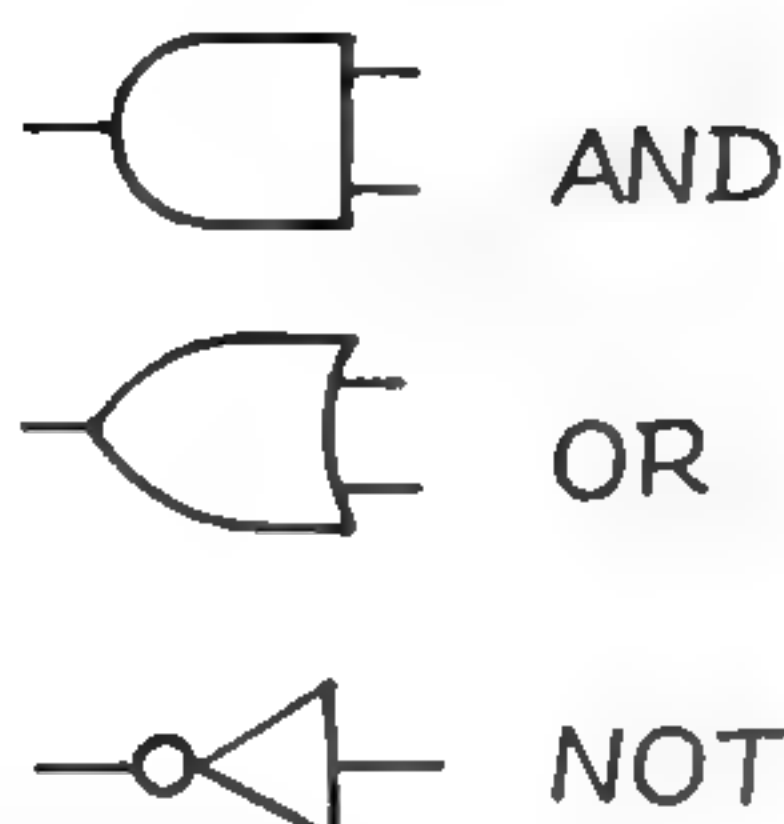
たとえばANDの場合を考えてみます。AとBの2入力の場合(2ビットと考える)、入力の状態は4通りとれます。それぞれの状態に対する出力Cの状態がANDの場合はこうなるのです。

真理値表

インプット		アウトプット
A	B	C
0	0	0
0	1	0
1	0	0
1	1	0



ロジックにはAND(アンド)、OR(オア)、NOT(ノット)などがあり、ロジック回路を構成するうえで不可欠のものです。ANDやORには最低2入力が必要です。

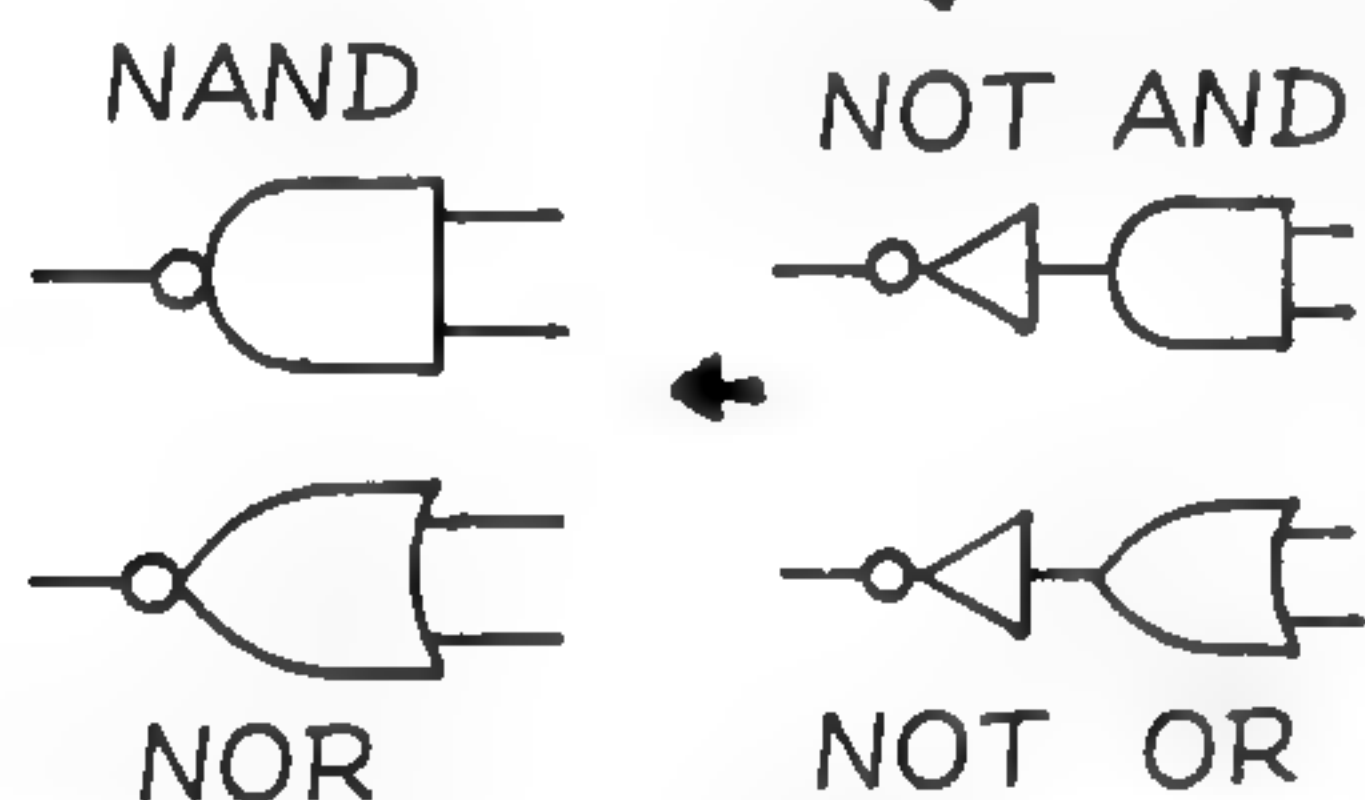


## ②ロジック用IC

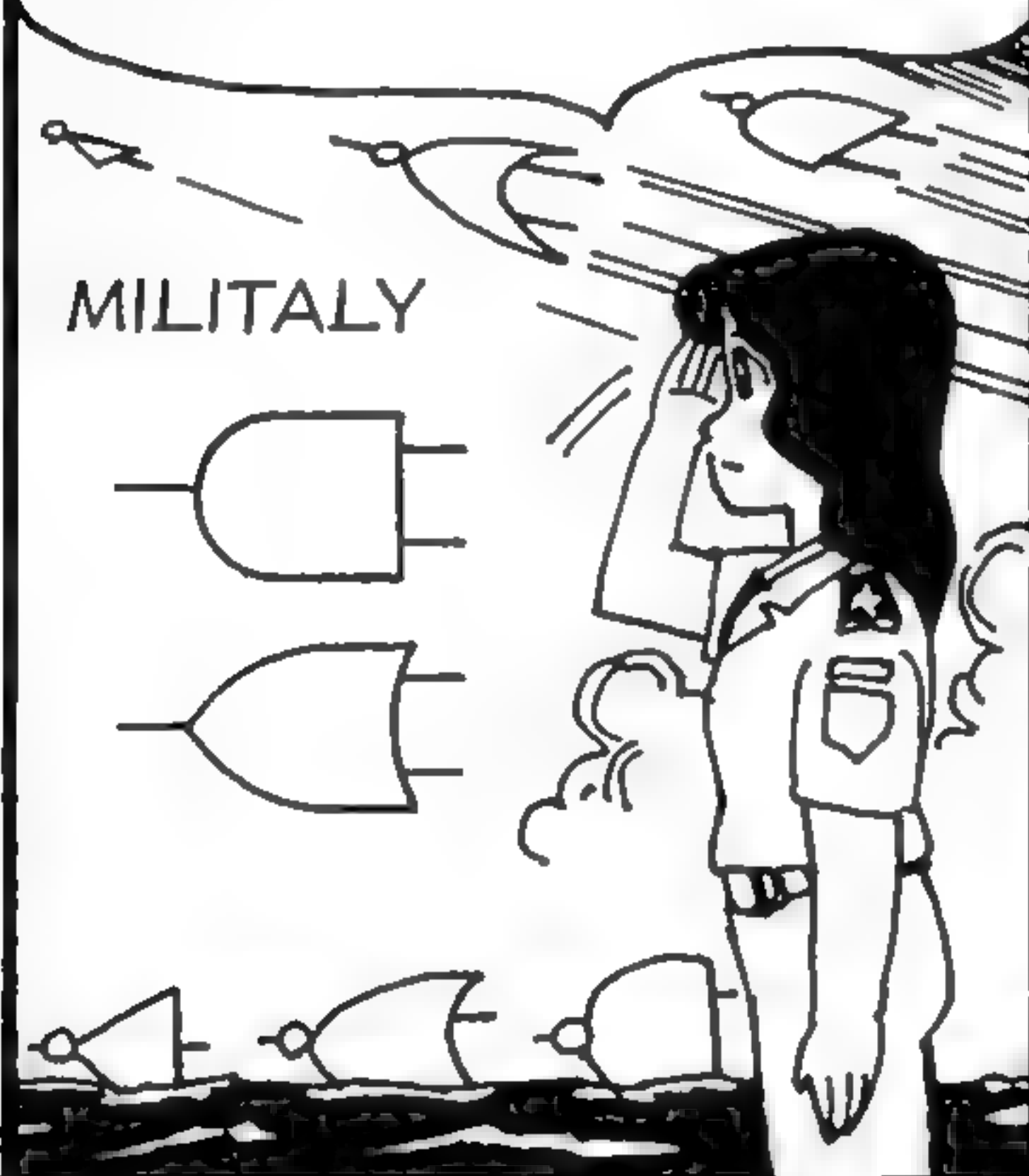
うまく考えられたシンボルマークです。



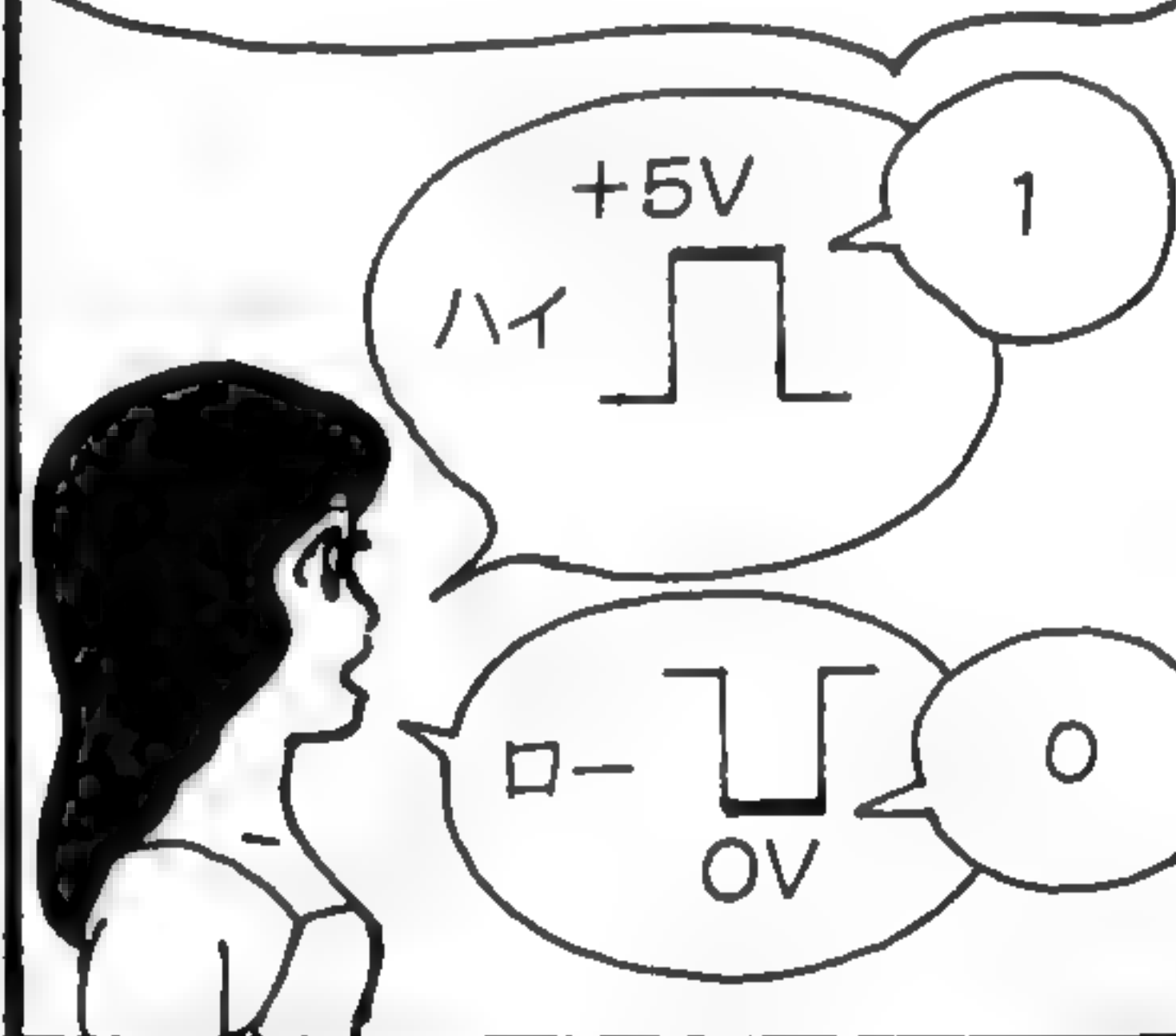
出力側に小さなマルがつくと否定を表わします。たとえばANDの先にマルのあるものはNOT・ANDとなりNAND(ナンド)と呼ばれます。ORの場合は同じやり方でNOR(ノア)となります。



また、ロジックを表わすシンボルで一番多く使われているのはアメリカのMIL規格のものです。



この表の1や0はそれぞれ電圧信号の有(5V)、無(0V)に対応しています。それで1はHIGH(ハイ)、0はLOW(ロー)と呼んでいます。



またNANDゲートでも出力が0の時の条件を強調しない時と1の時に強調したい時にはシンボルが変わります。

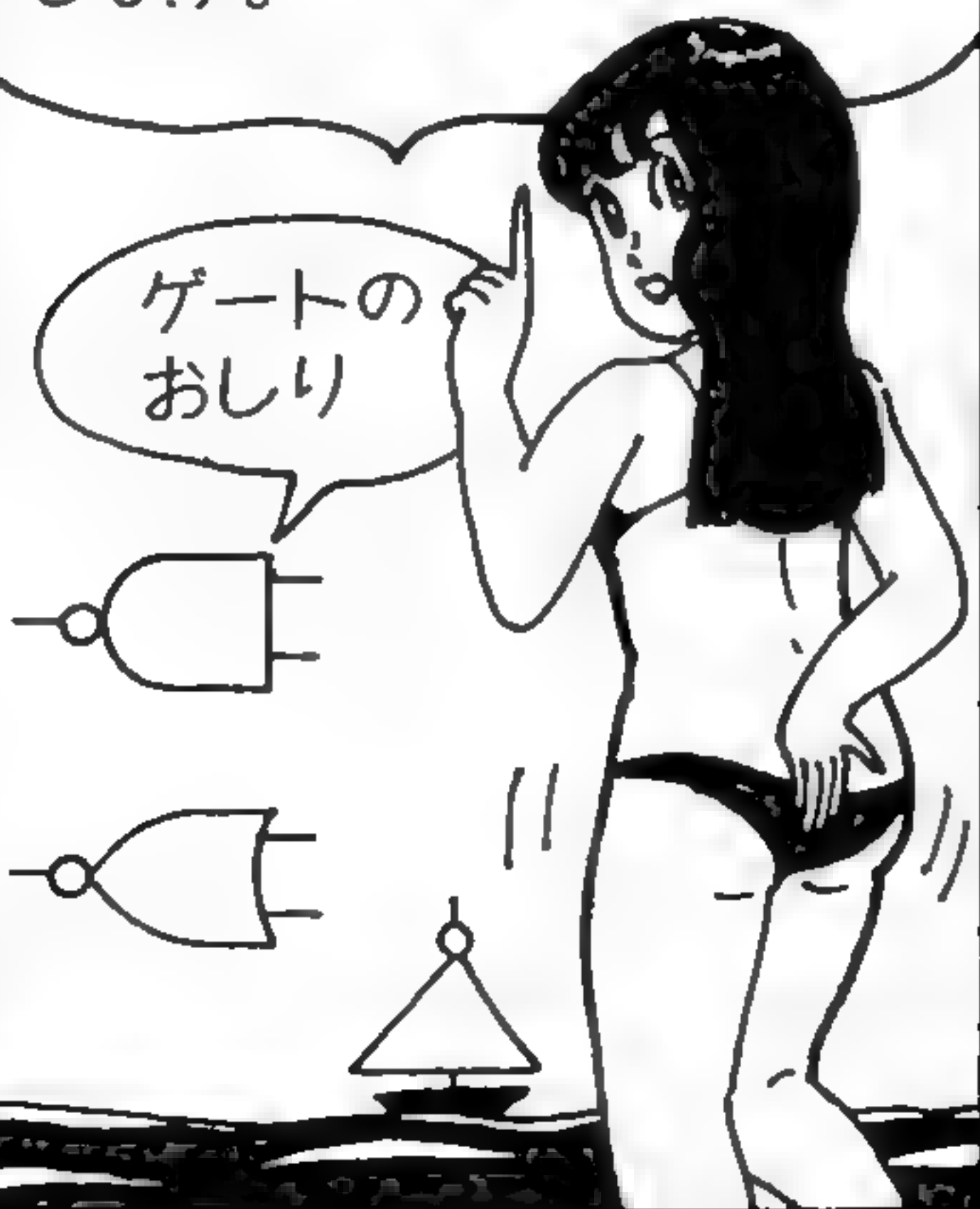


入力がABとも1で出力が0となる。



入力がABいずれか0のとき出力が1となる

ゲートのおしりにも意味があって、まっすぐはAND、弧になっているものはORを示します。



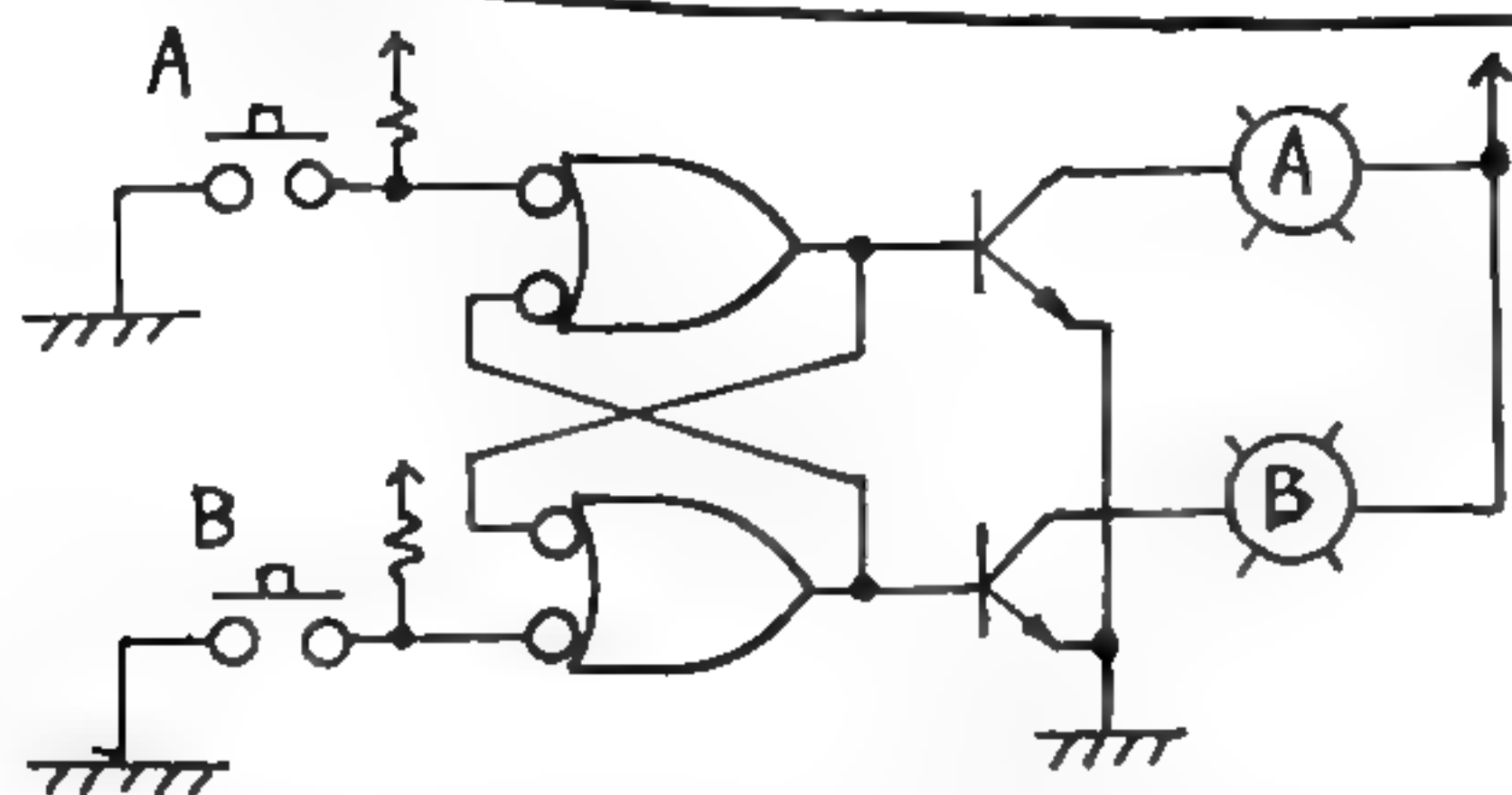
これはNANDゲートの動作を表わす表です。むずかしくいうと真理値表です。

インプット		アウト
A	B	C
0	0	1
0	1	1
1	0	1
1	1	0

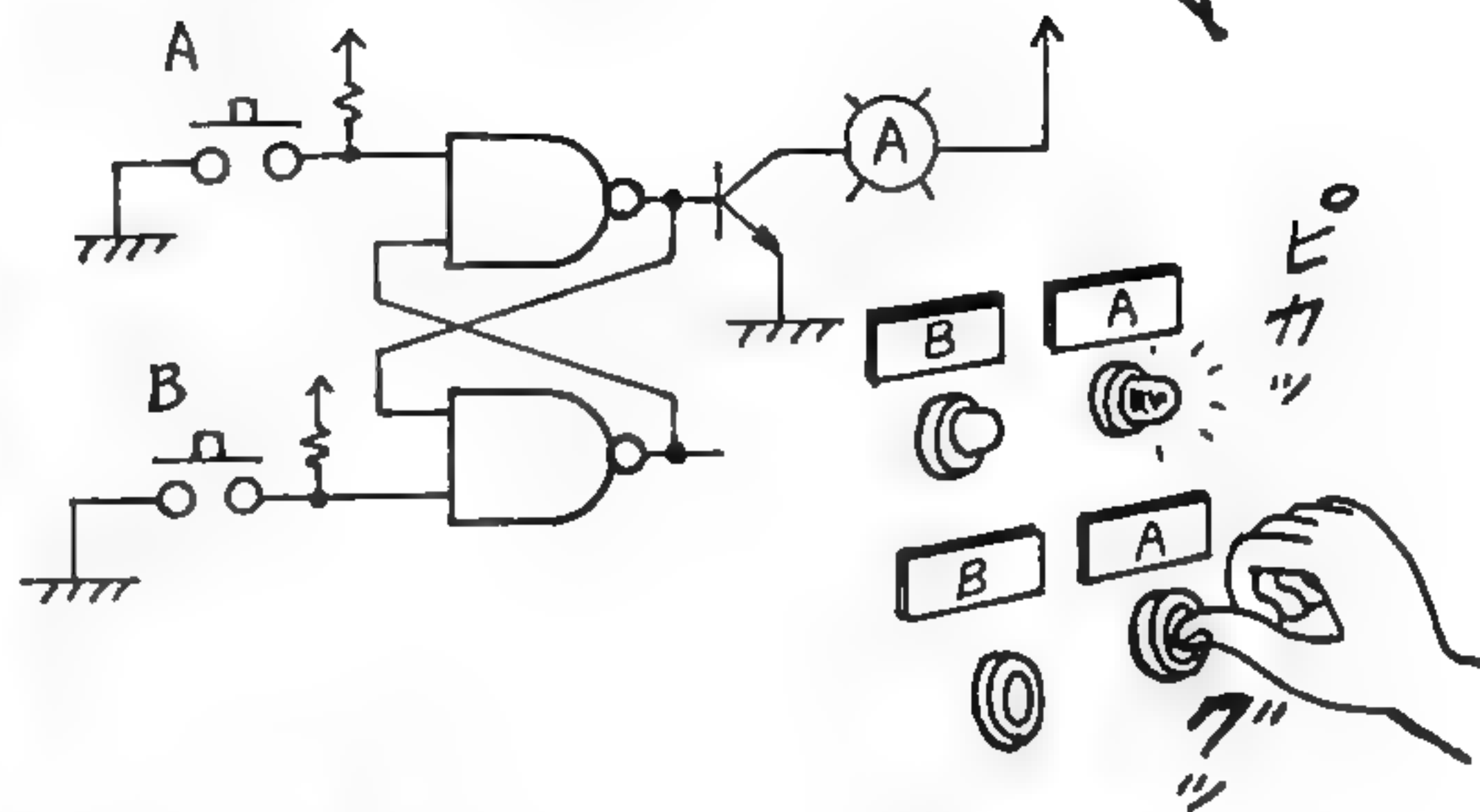
シンボル化



ところがこの回路はスイッチを押した時、つまり入力が'0'になった時、出力が'1'になってその状態が維持されますのでこう表現した方がわかりやすくなります。



これはNANDゲート2つを使ってできるフリップ・フロップという電子的な自己保持回路です。



テキサスインスツルメンツ社のTTLロジックICは応用製品のデジタル化と共に一世を風靡した感がありましたね。



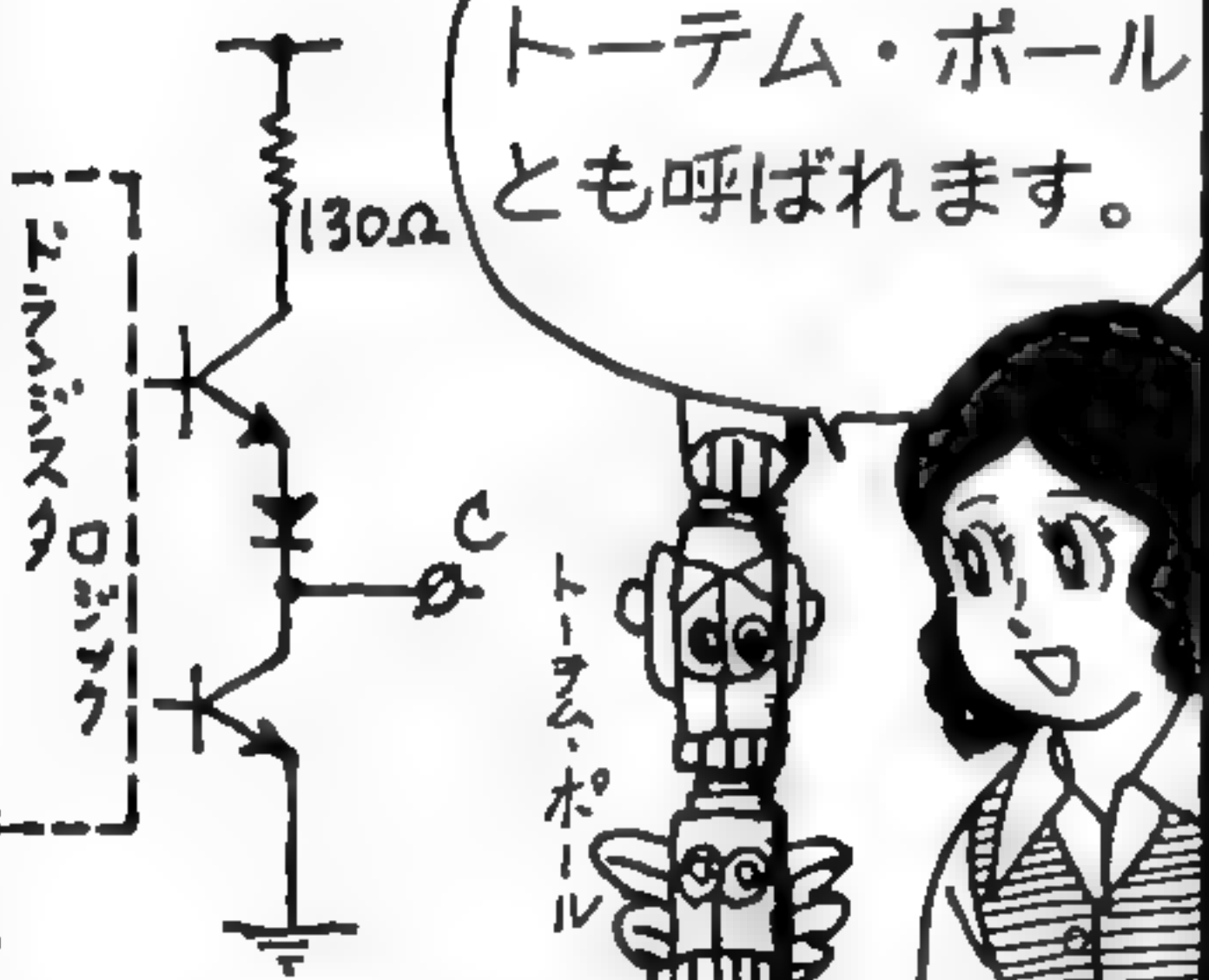
ICの初期のころは、この比較的小さな回路のゲートから製品化されたんです。



これらのICはゲートとも呼ばれています。

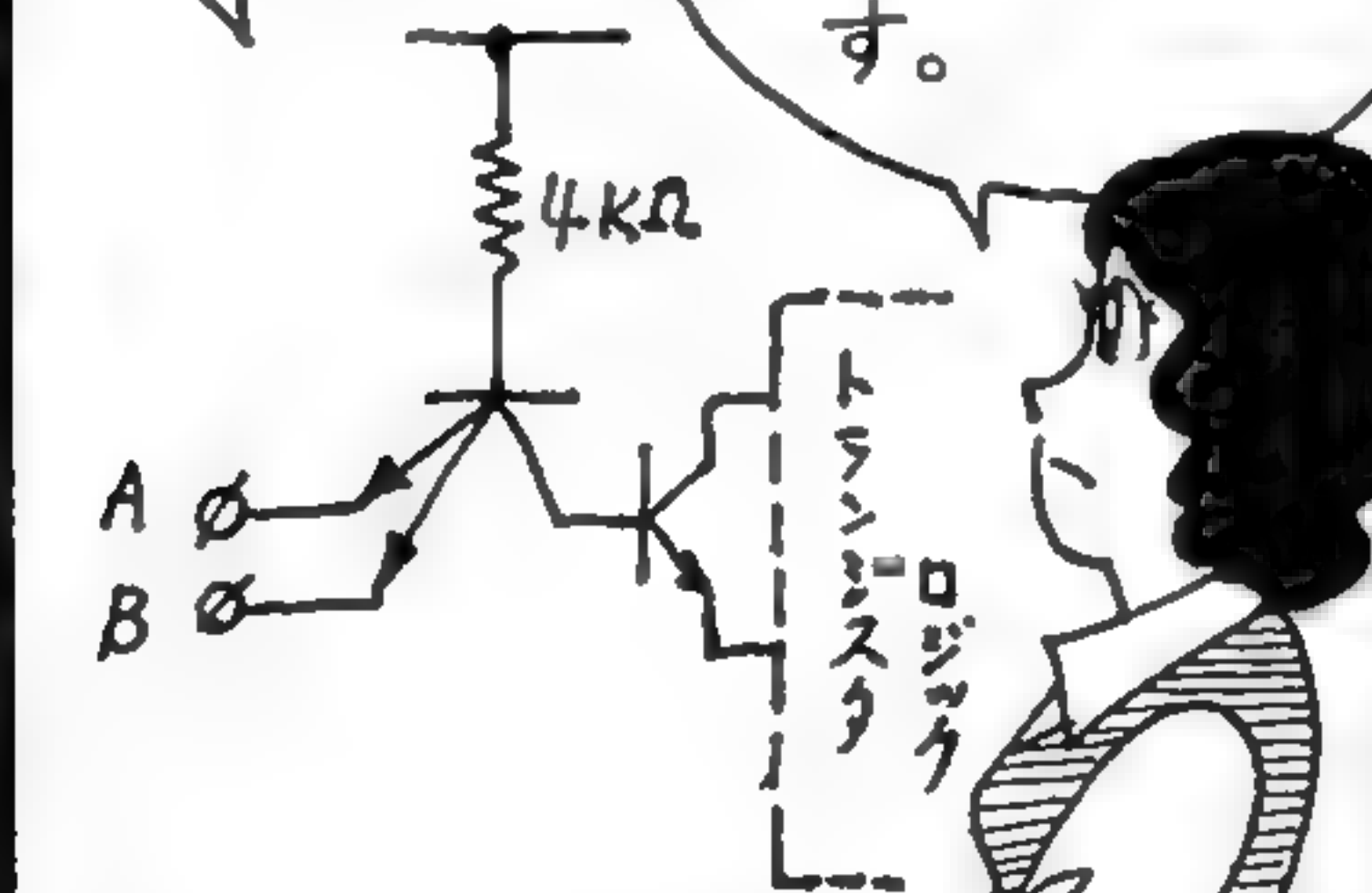


これはゲートの出力回路の部分でトータム・ポールとも呼ばれます。



これはゲートの入力回路です。

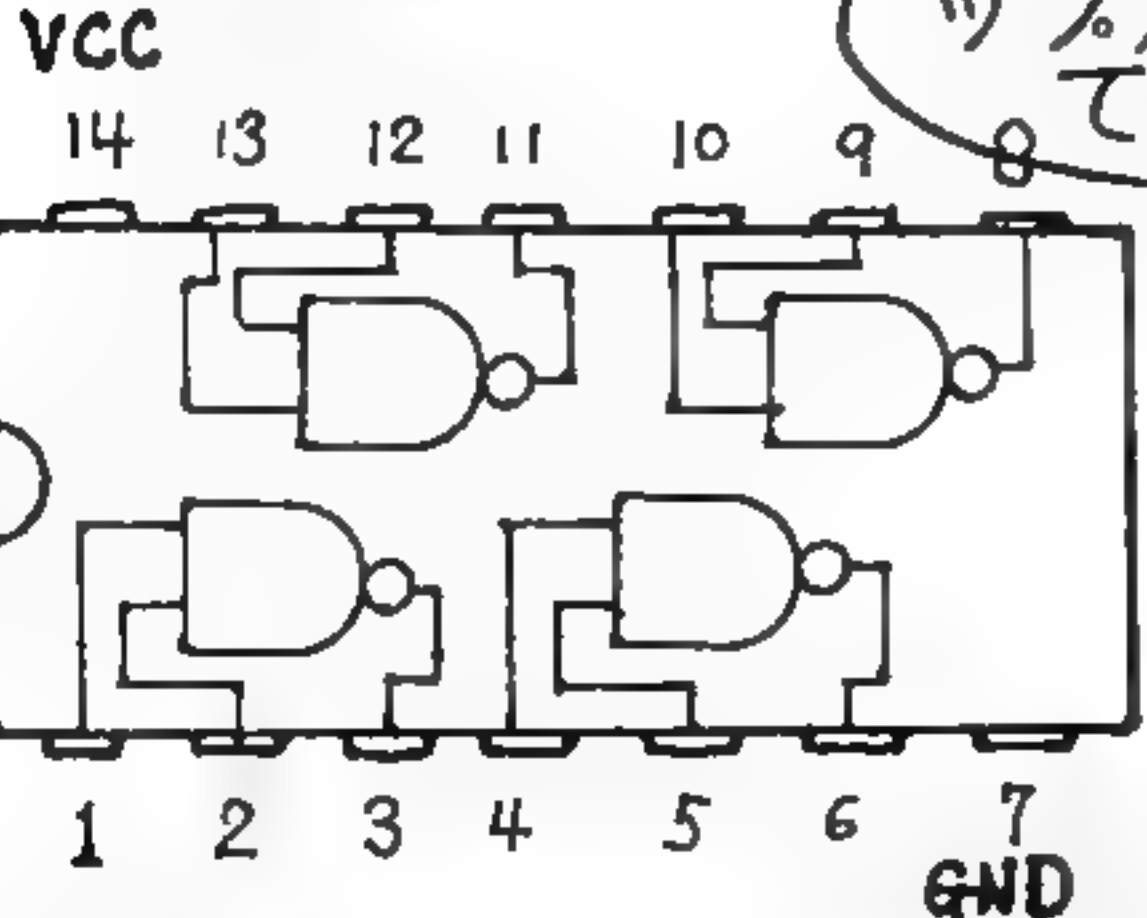
文字通りTTLはトランジスタ回路で論理を構成しています。



TTL(ティティエル)というのはトランジスタ・トランジスタ・ロジックの略です。



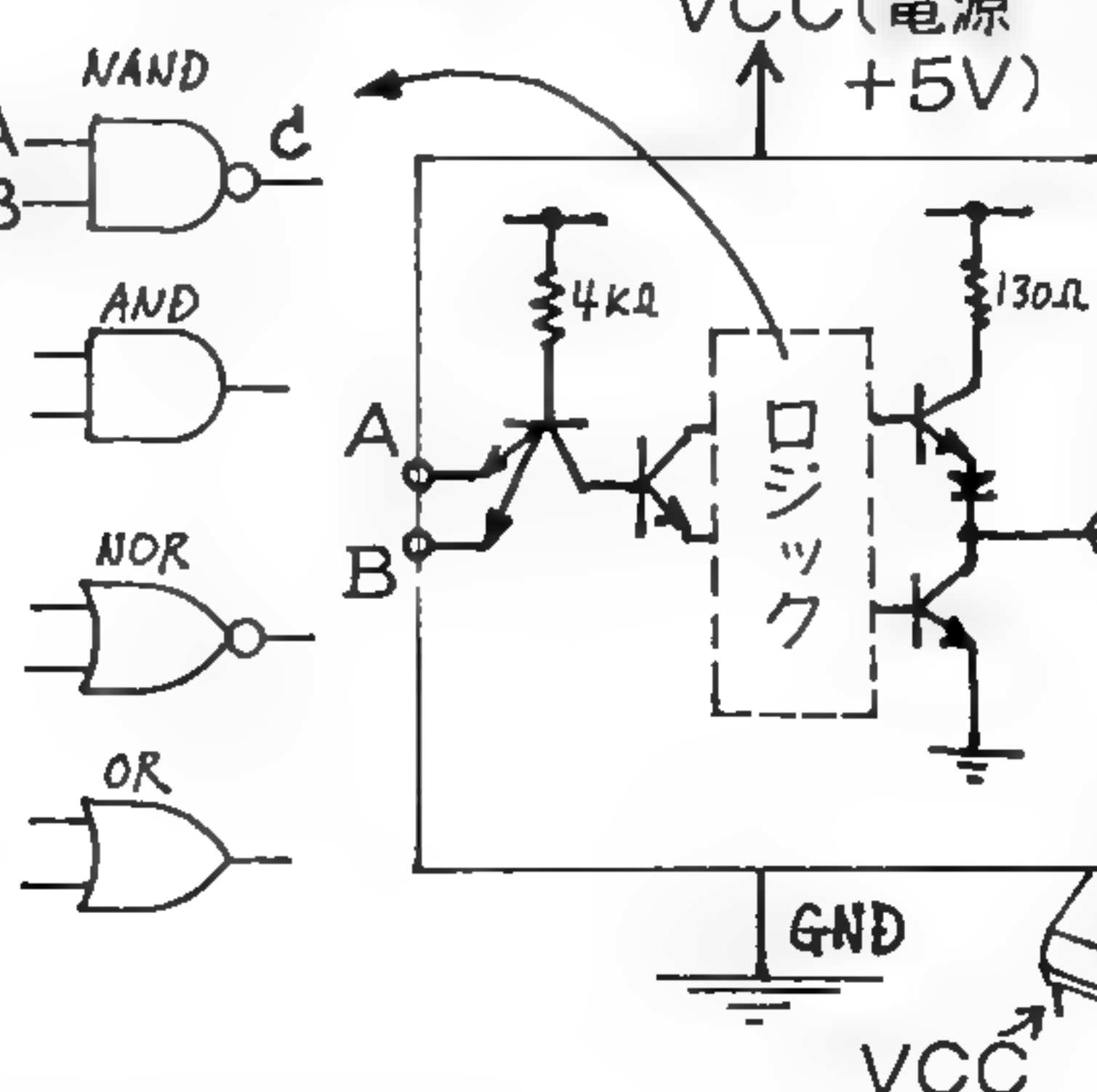
テキサスのIC SN7400



カタログなどに載っているこのマークはICのくぼみでピンの番号を知る目印です。



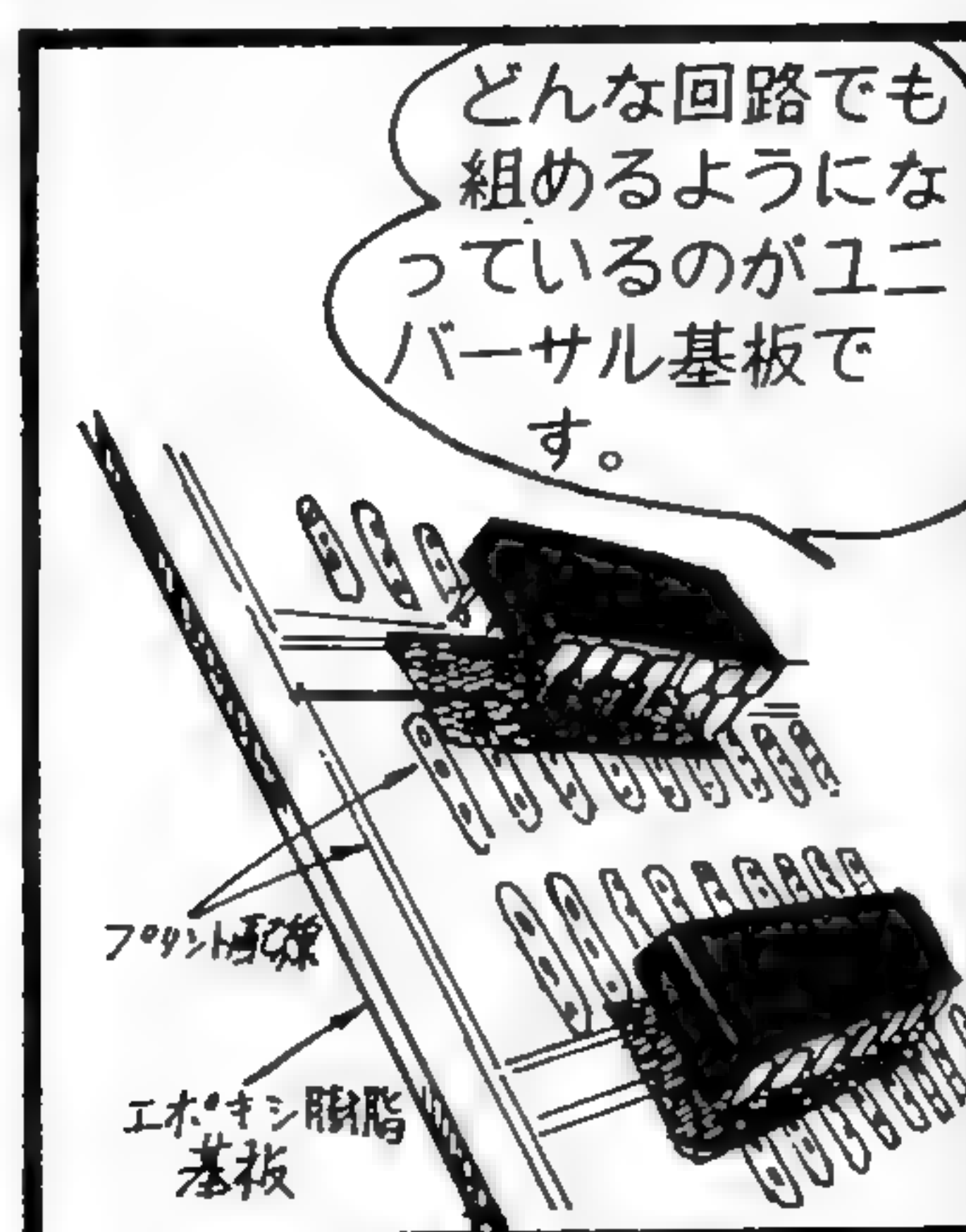
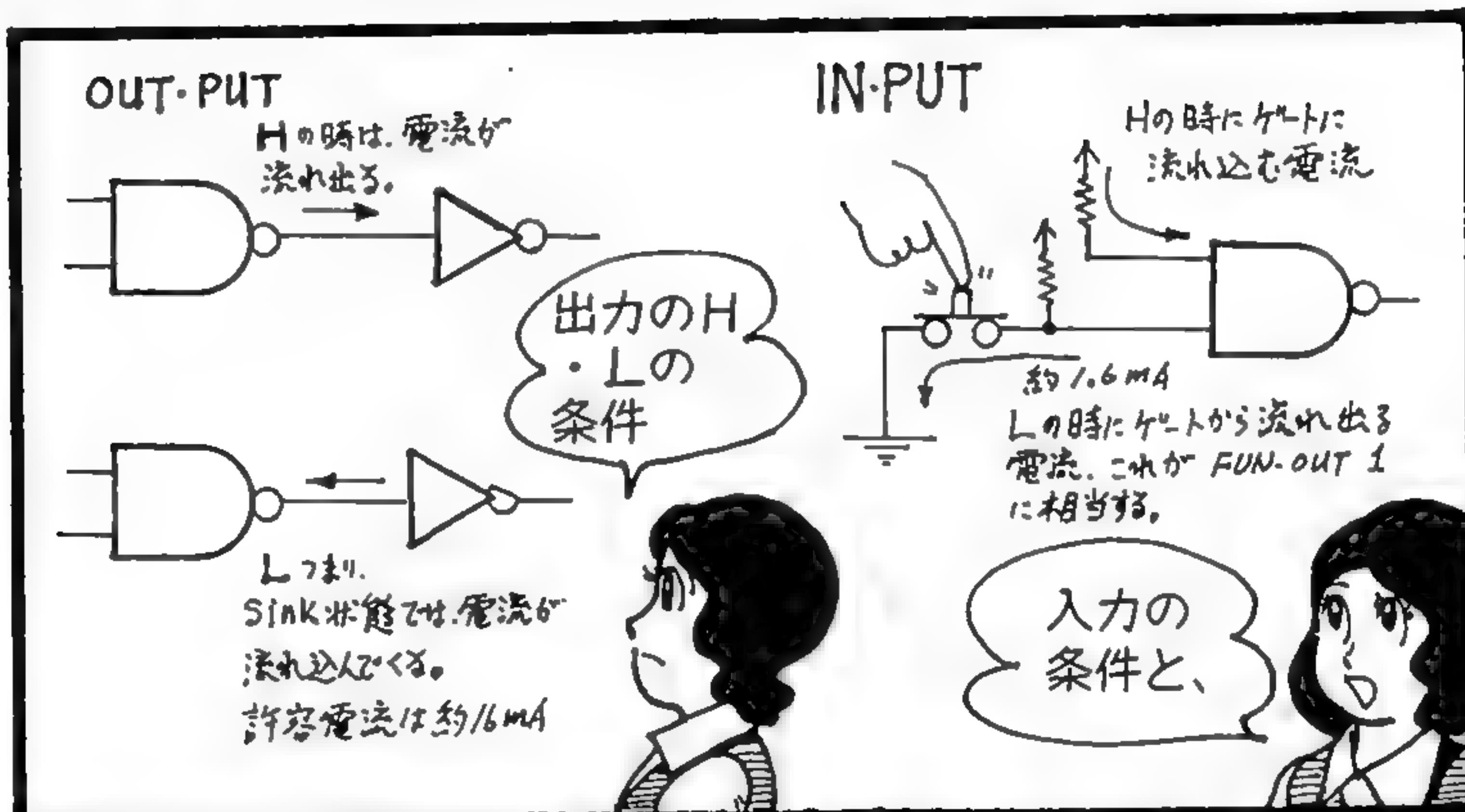
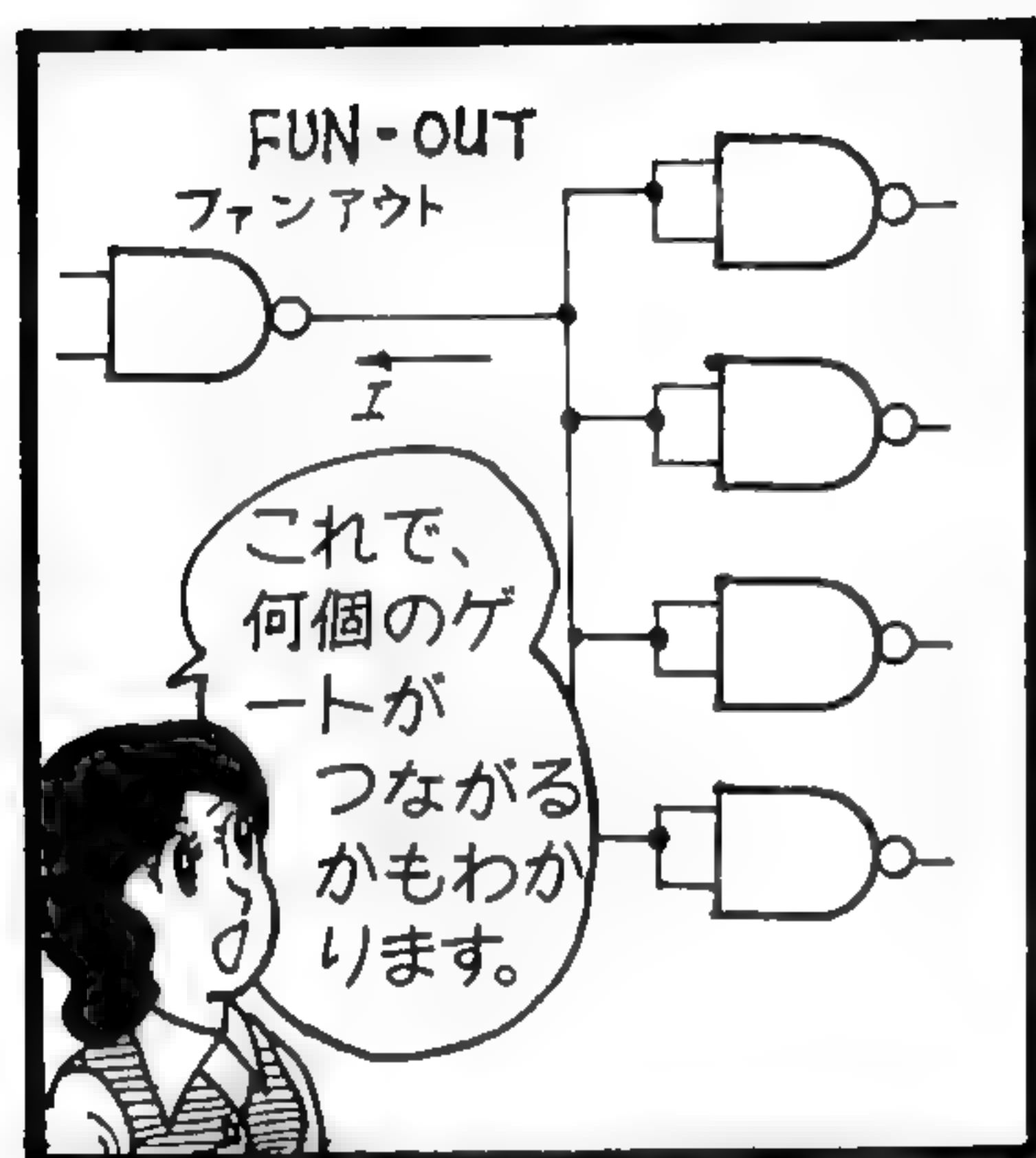
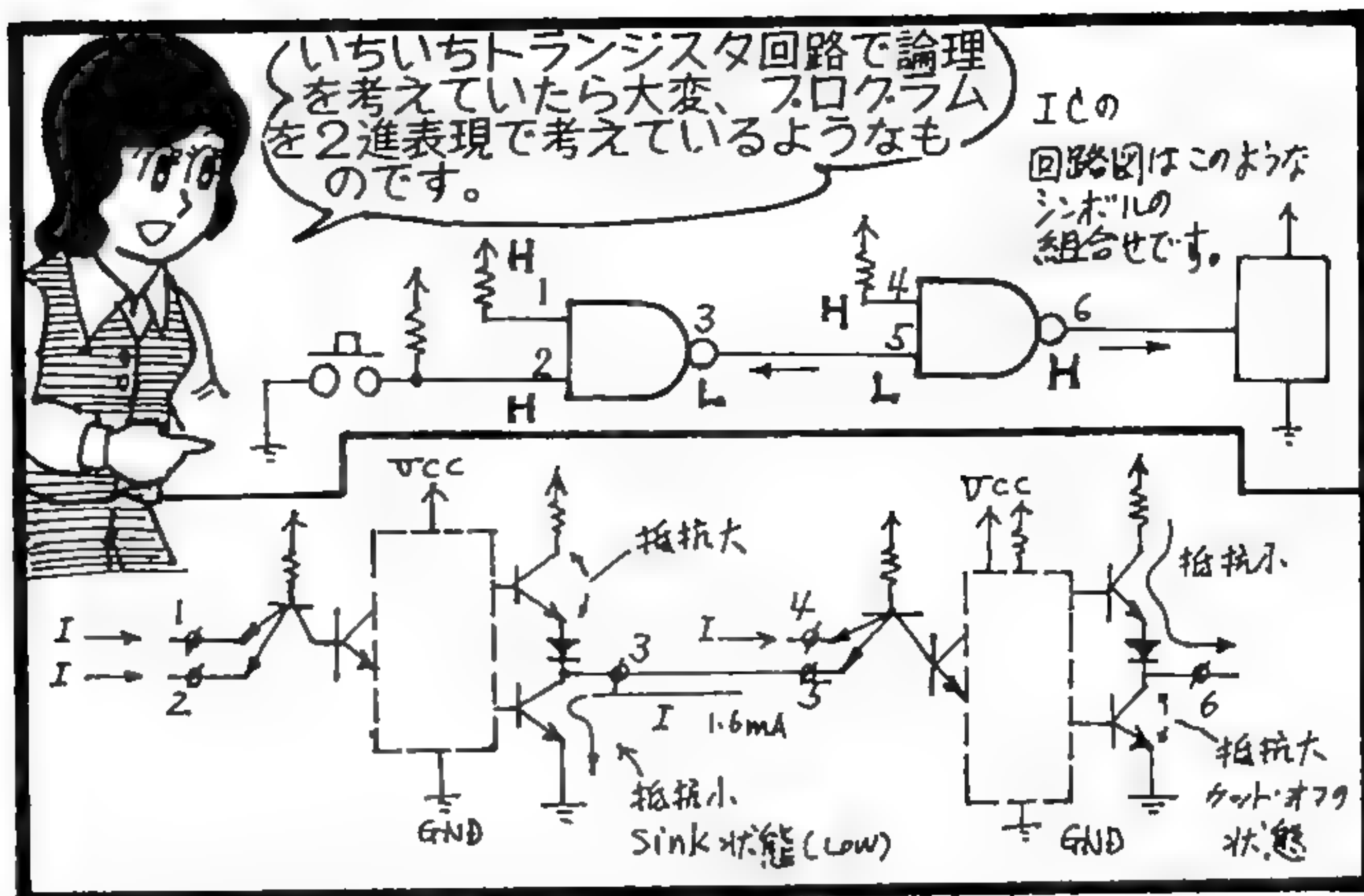
ここで決まる



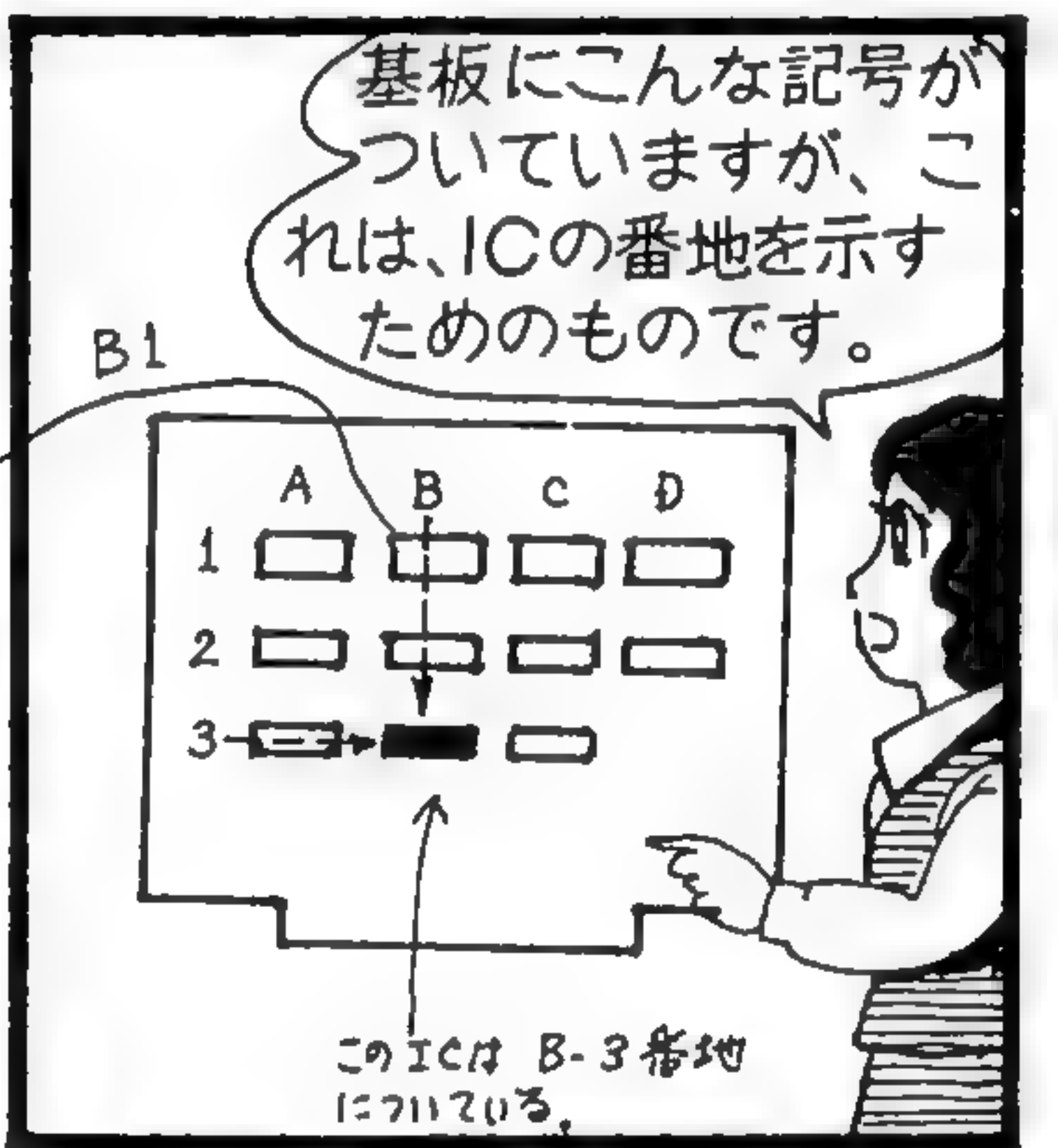
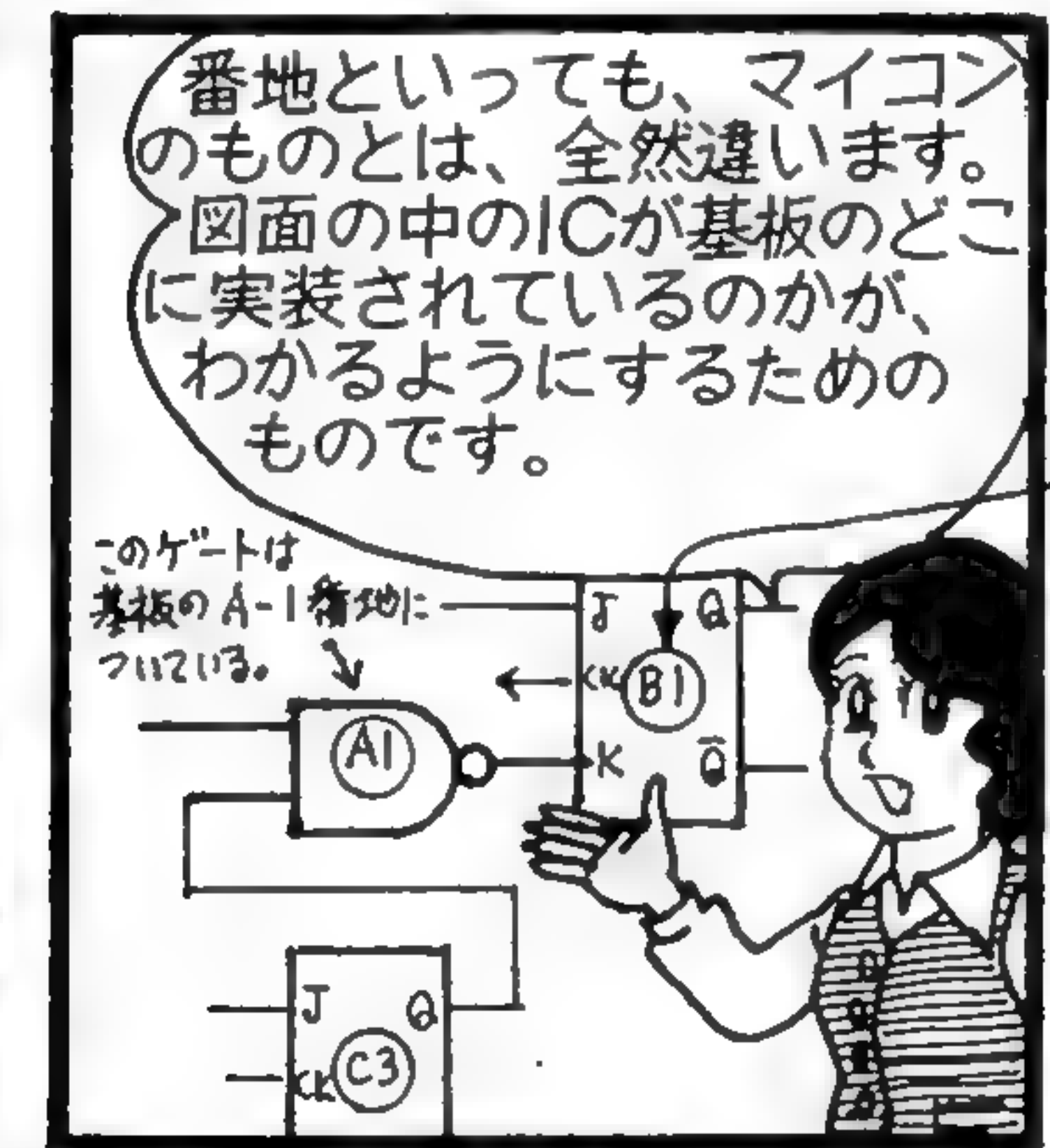
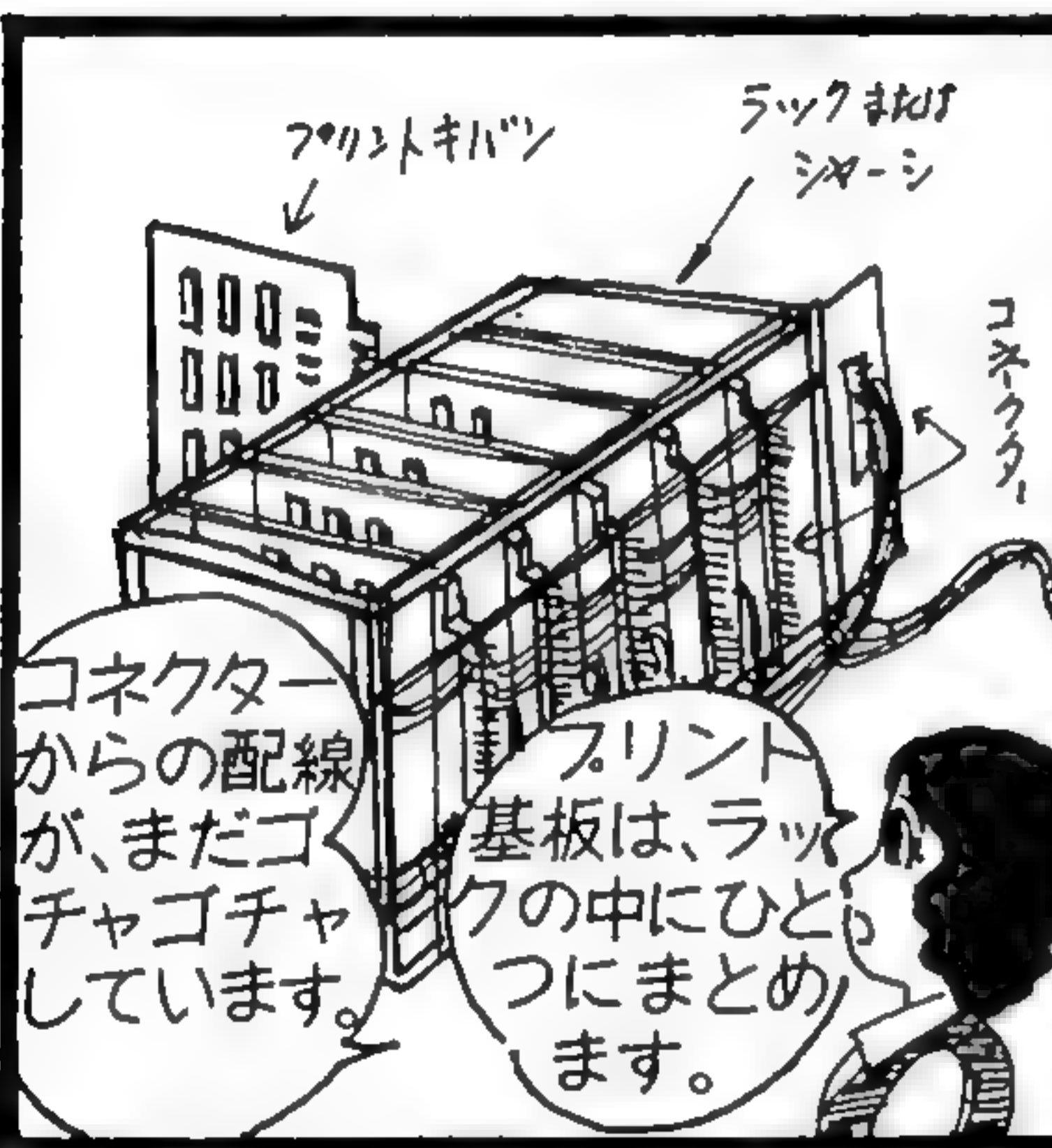
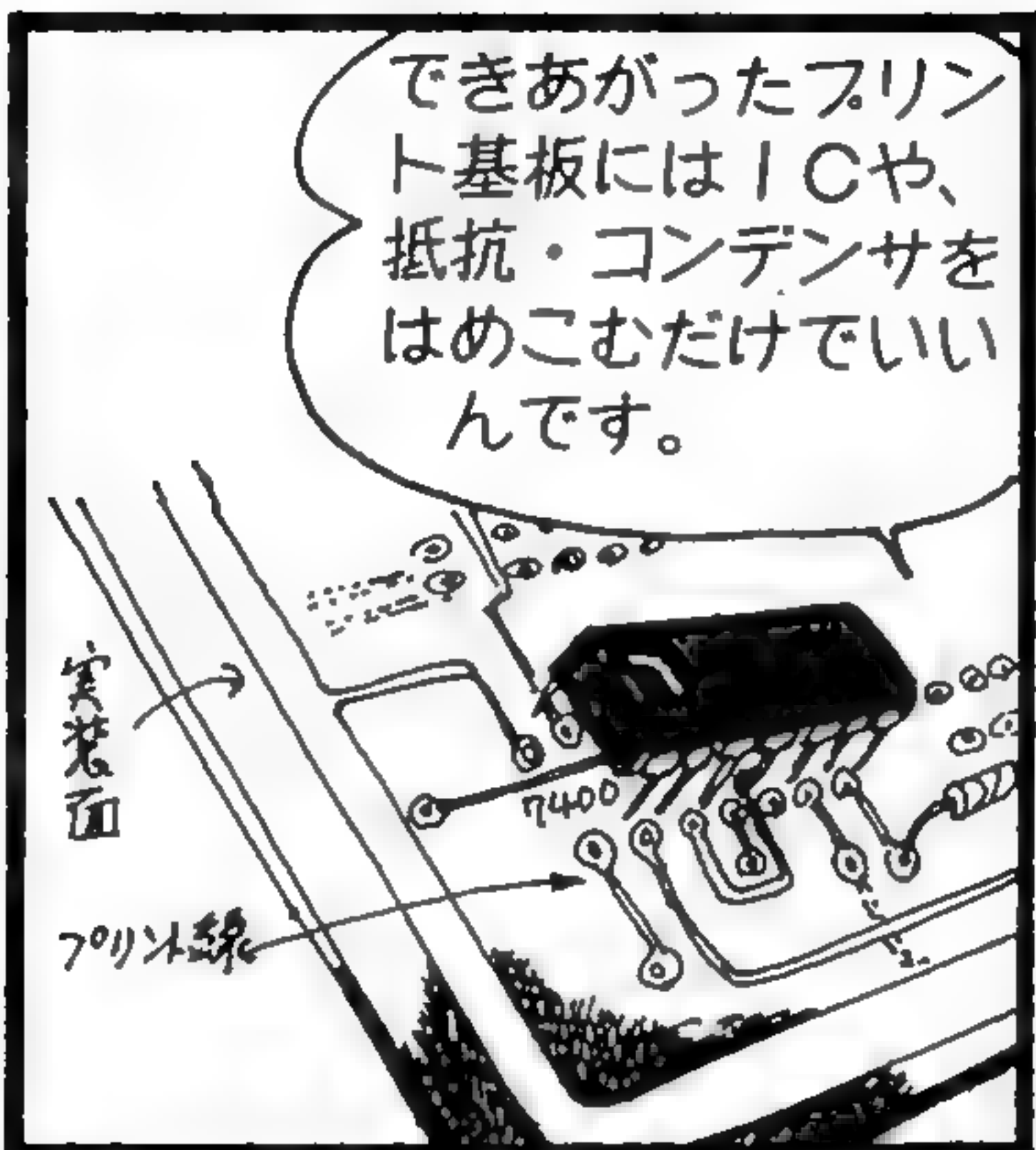
つまりロジックの部分の組み合わせでいろんなゲートができるんです。





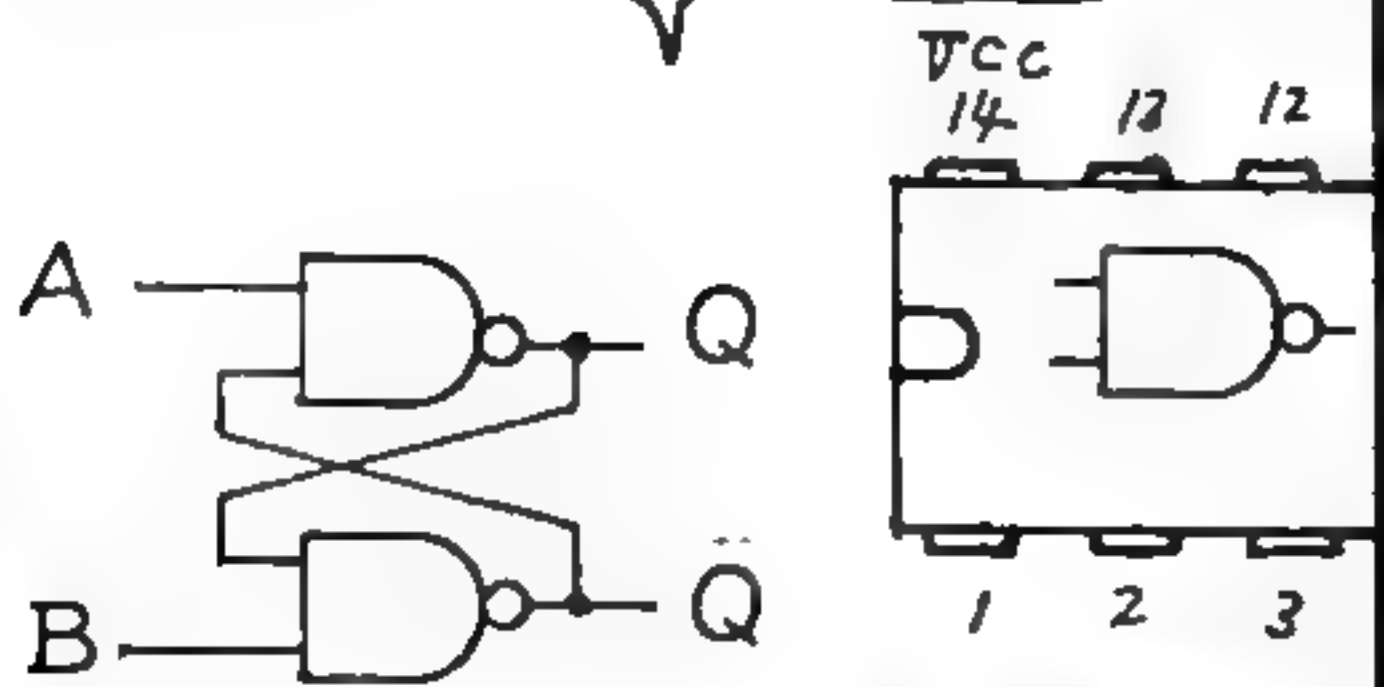






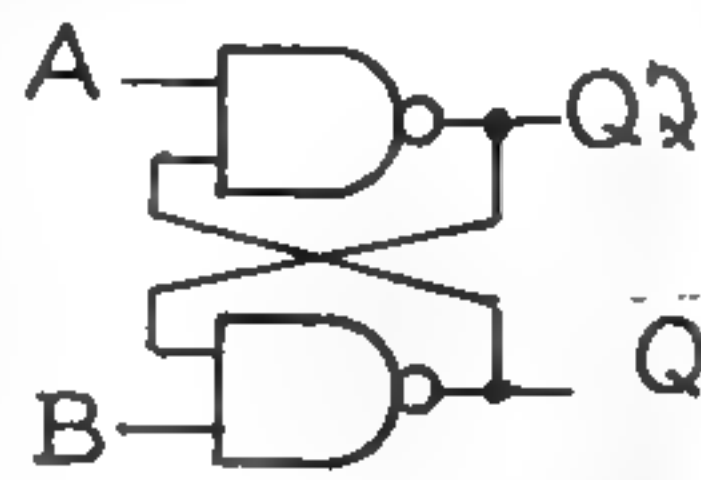


必要ならば NANDゲート2個で作ります。これでこのFFが実現されたこととなります。



こんな形をしたFFはICパッケージの製品としては発売されていません。

フリップフロップ回路には情報をためておける。

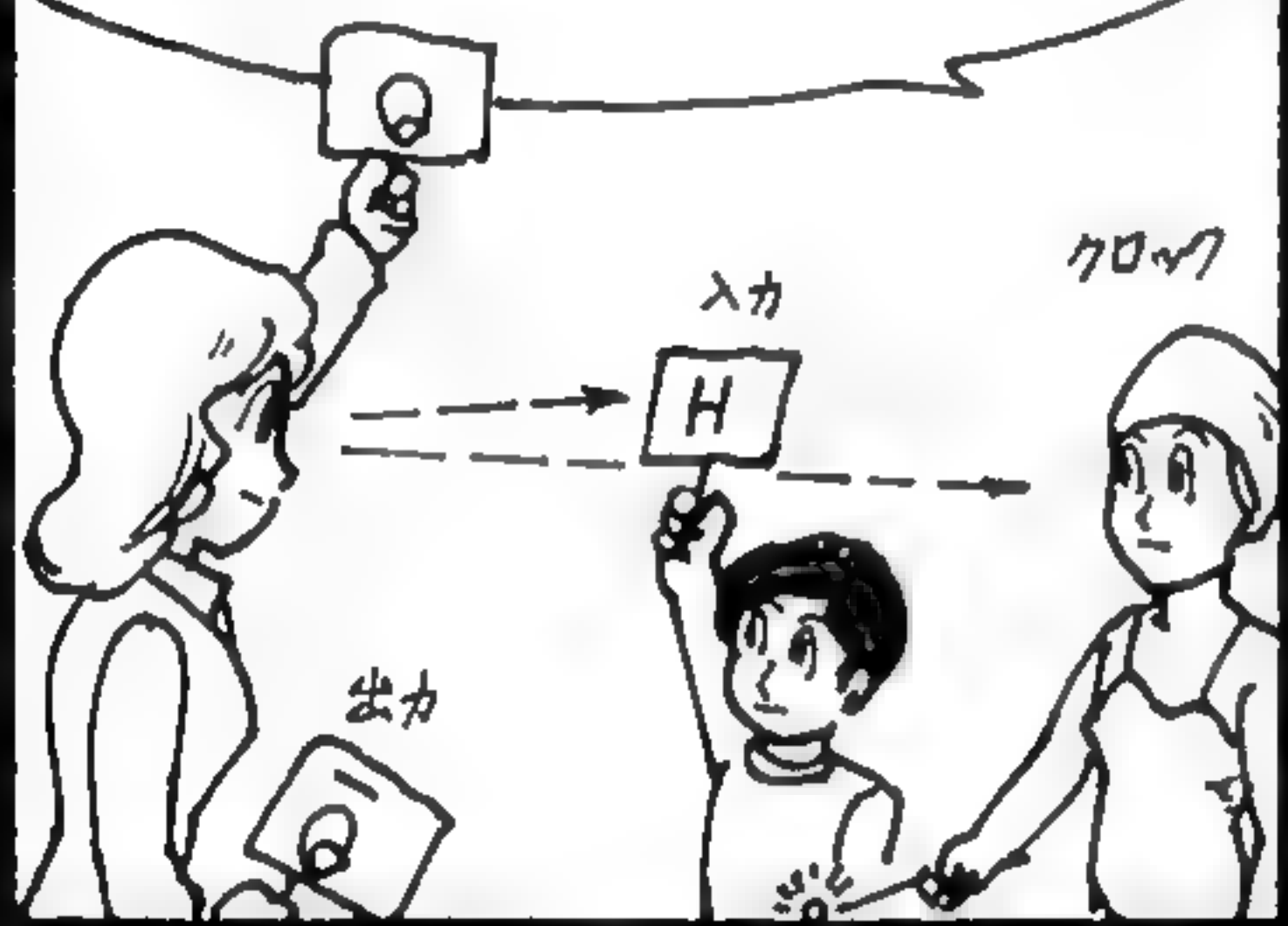


### ③フリップフロップ

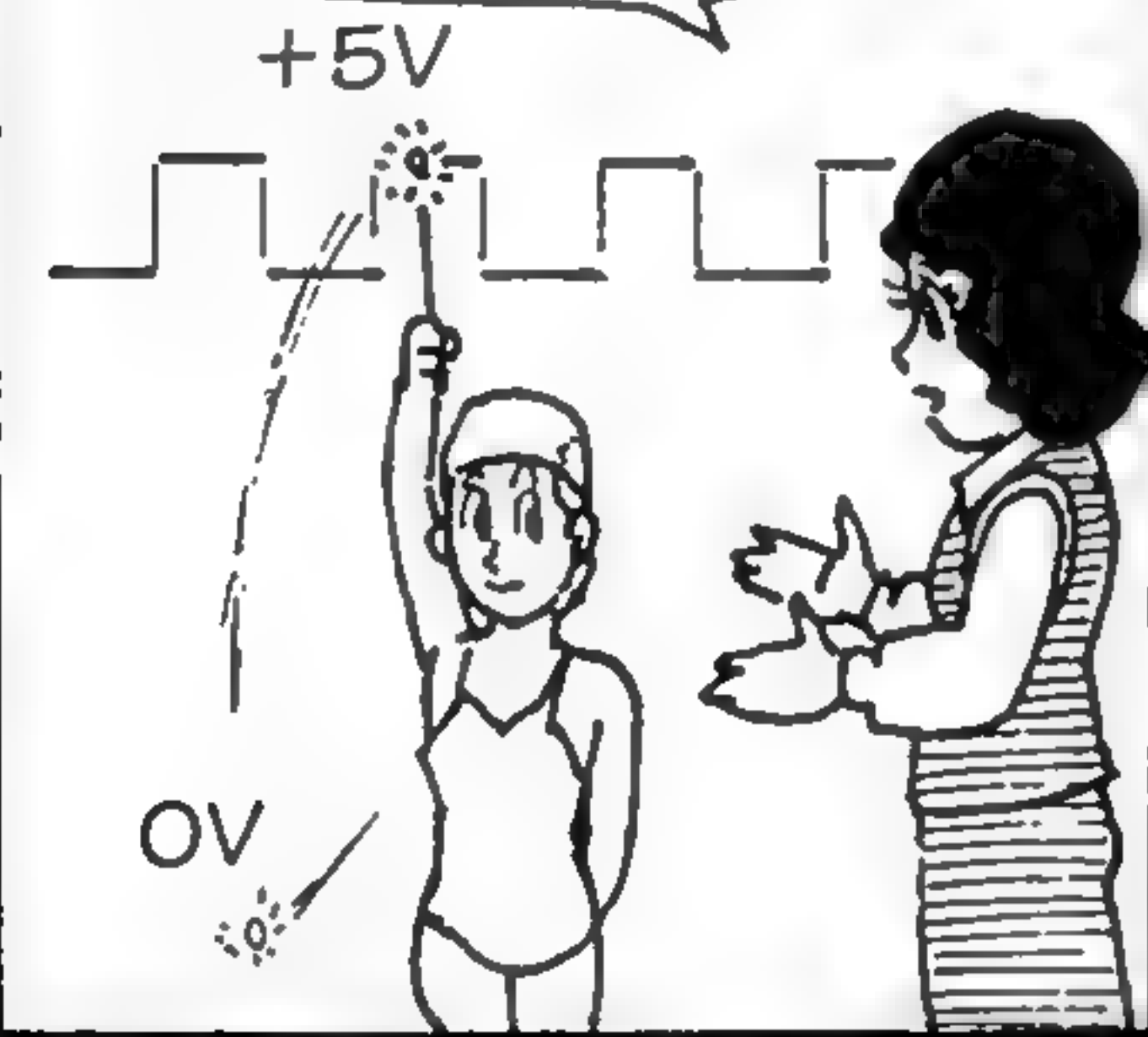
キッタン  
ハッタン



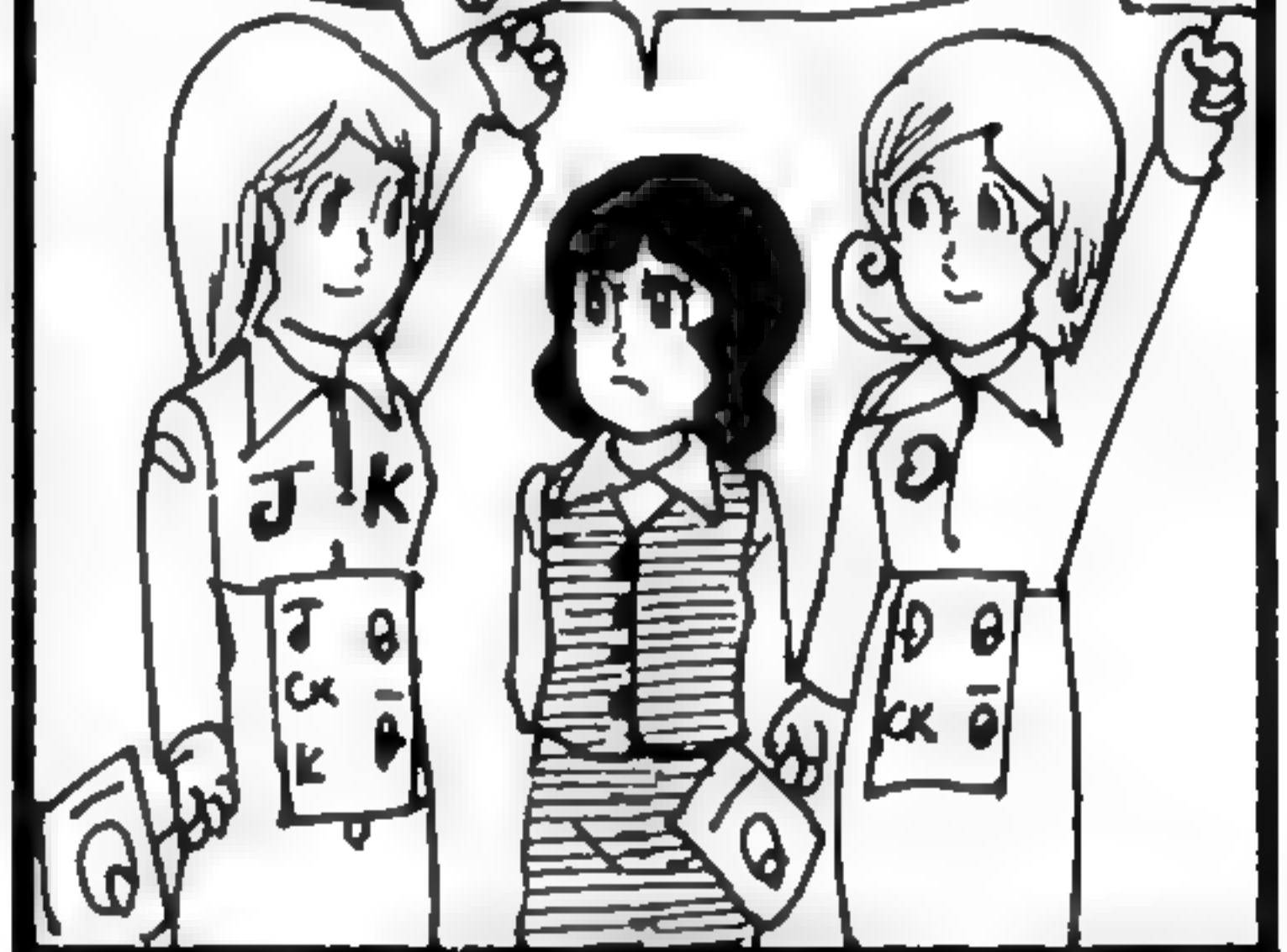
クロックパルスの立上り、または、立下りのエッジによって、出力が変化するものです。



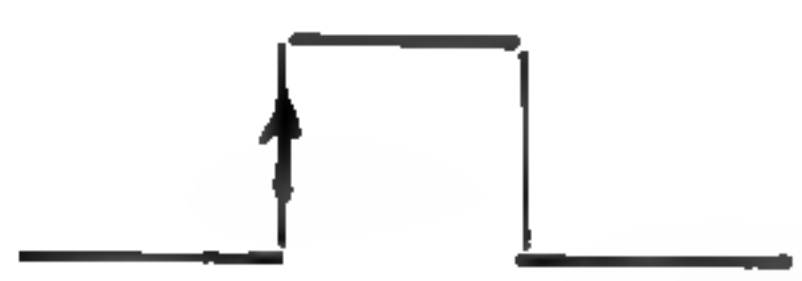
この2つはクロックと同期をとるように考えられたものです。



パッケージに入っているFFで代表的なものはエッジトリガー型J KとD型の2種類です。



DタイプのFFは正エッジのトリガーを持っています。



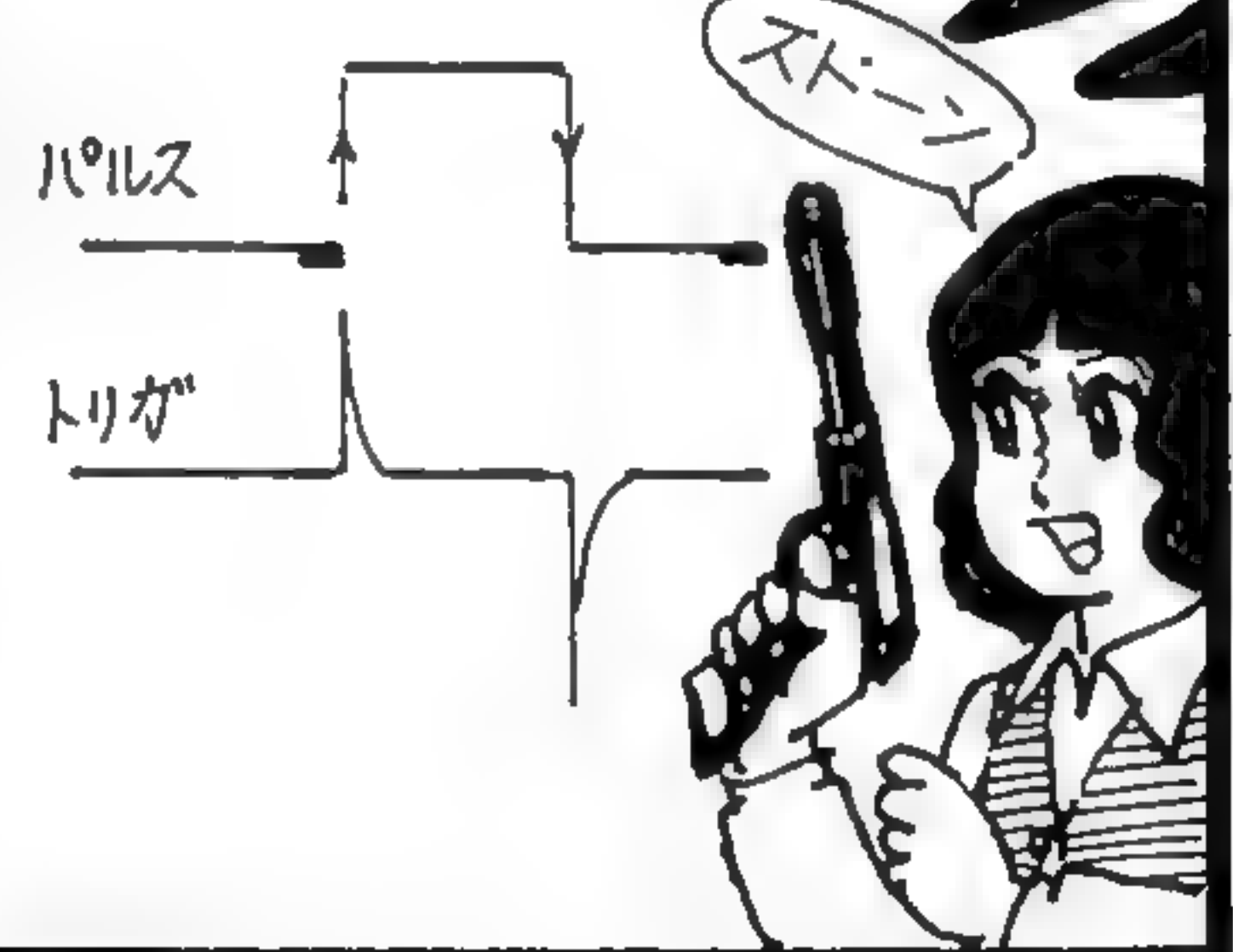
↑  
立上りで入力に対する出力が現われ保持されます。

J KタイプのFFは負エッジのトリガーを持ち、



↑  
立下りで入力に対する出力が現われ保持されます。

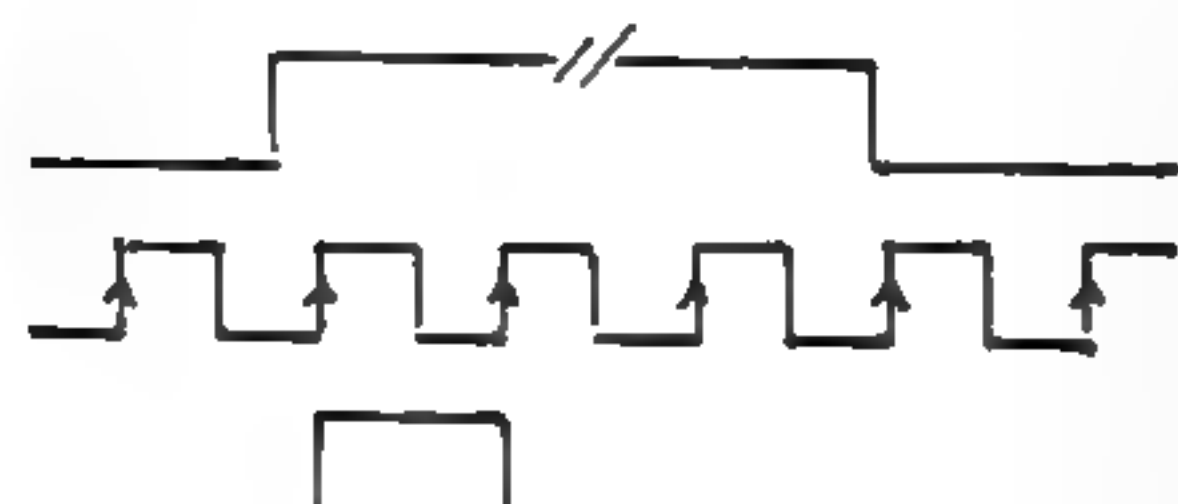
この、入力に対して出力を変化させる引き金になるものをトリガーといいます。



また、そのさい入力パルスの最初または終わりのどちらかで出すことも可能です。(もちろん、回路は違います)



たとえばDタイプのFFを使って不確定長のパルスをクロック1周期長のパルスにけずることができます。



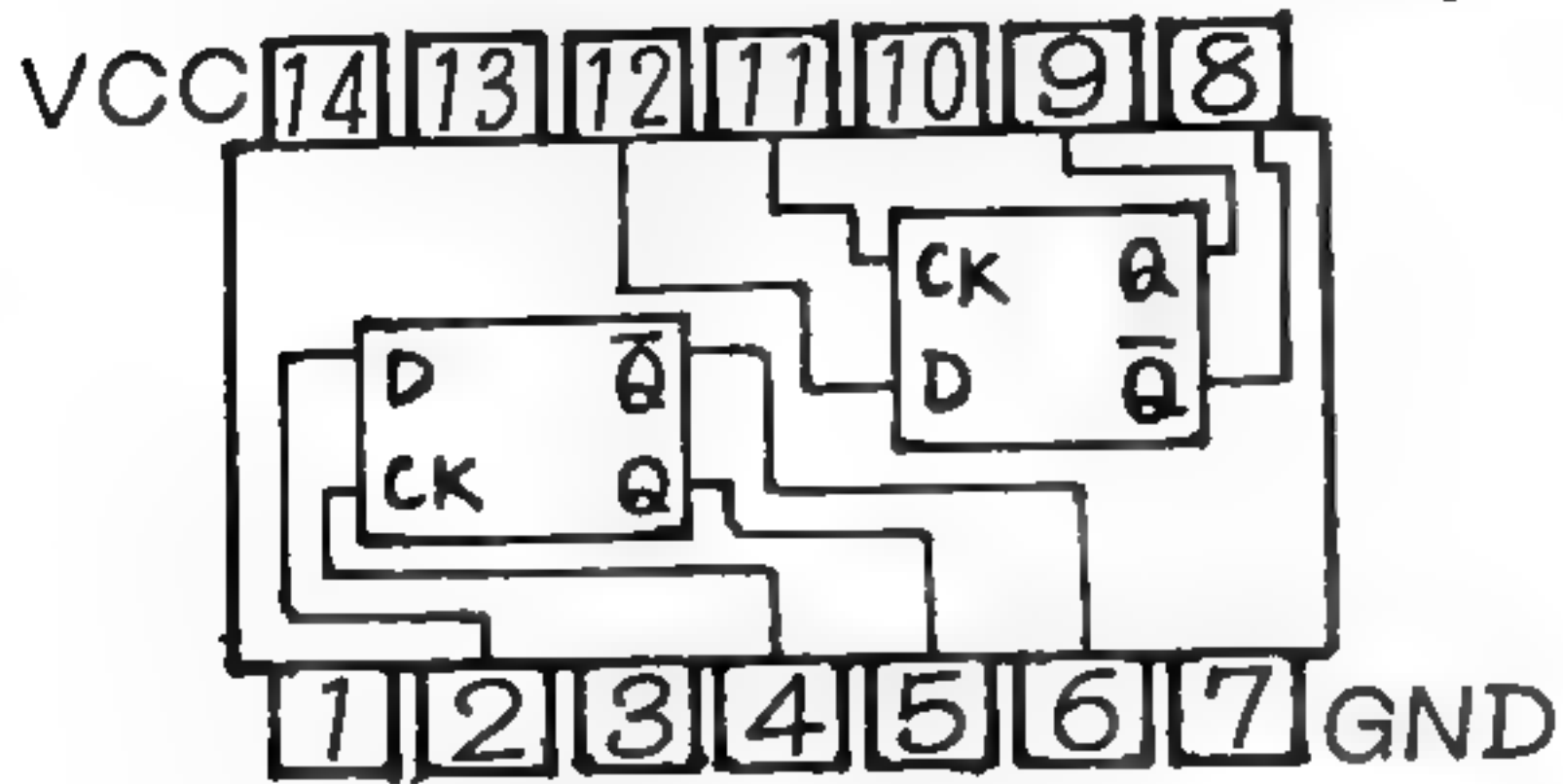
これらのエッジトリガー方式のFFはクロック1周期に1回しか入力値を出力する機会がありませんから、かえって応用がいろいろ効きます。またノイズにも強いのです。



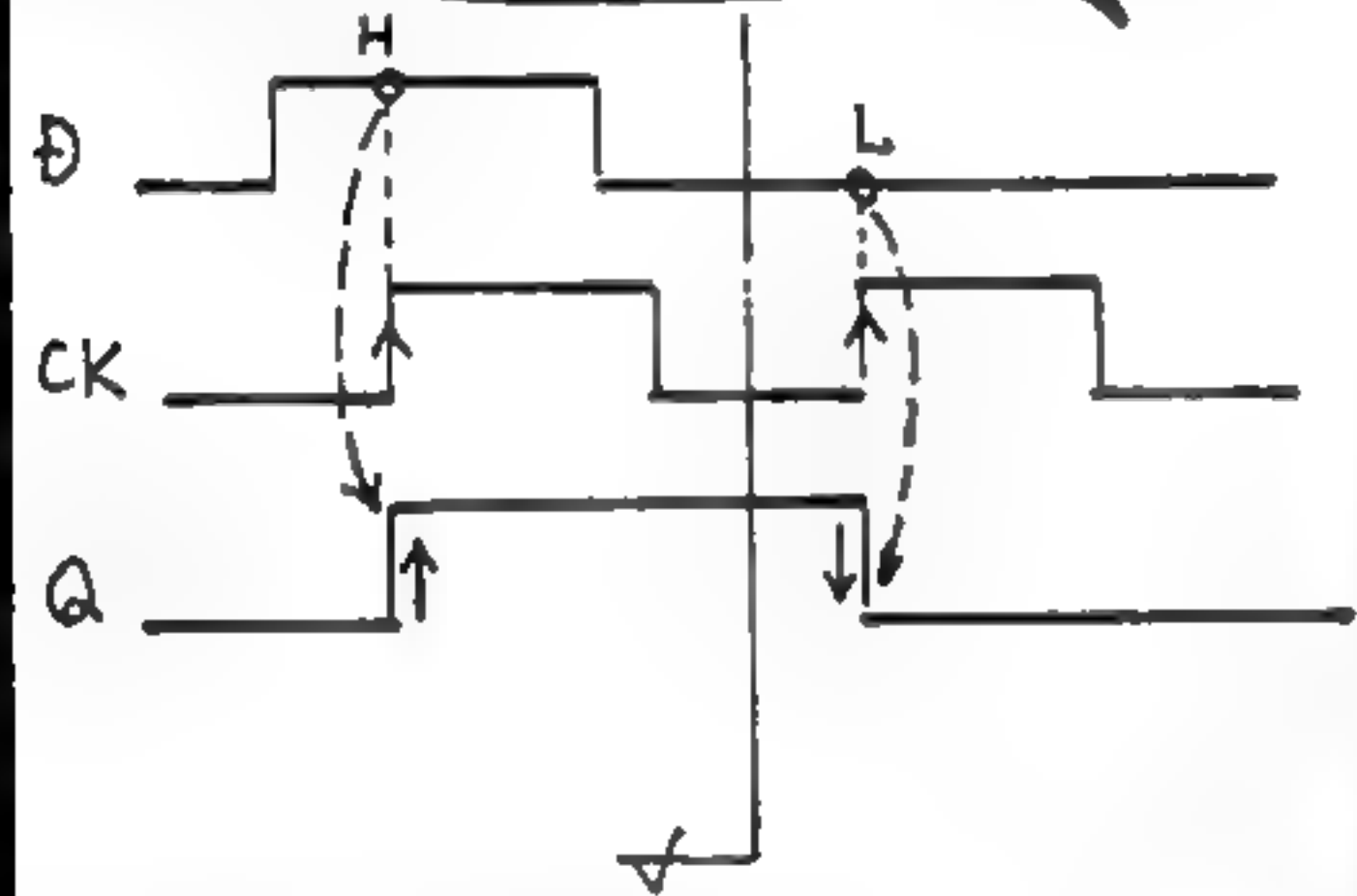
↑  
立上りエッジで使用する時  
このような印をつけることがあります。



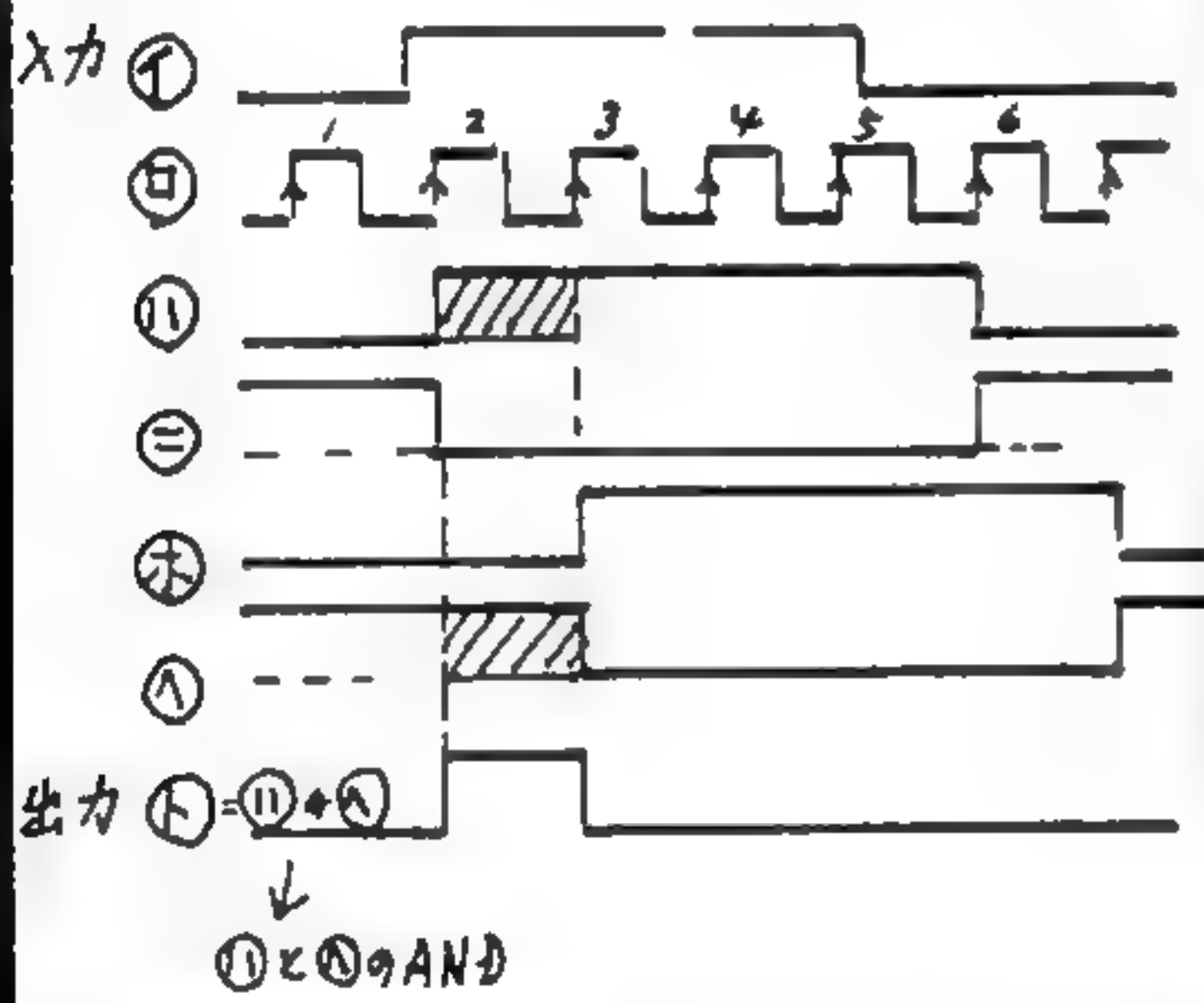
パッケージにはふつう2個のFFが入っています。どのピンがどの信号かはこのようなトップビューがカタログに出ているのでわかります。



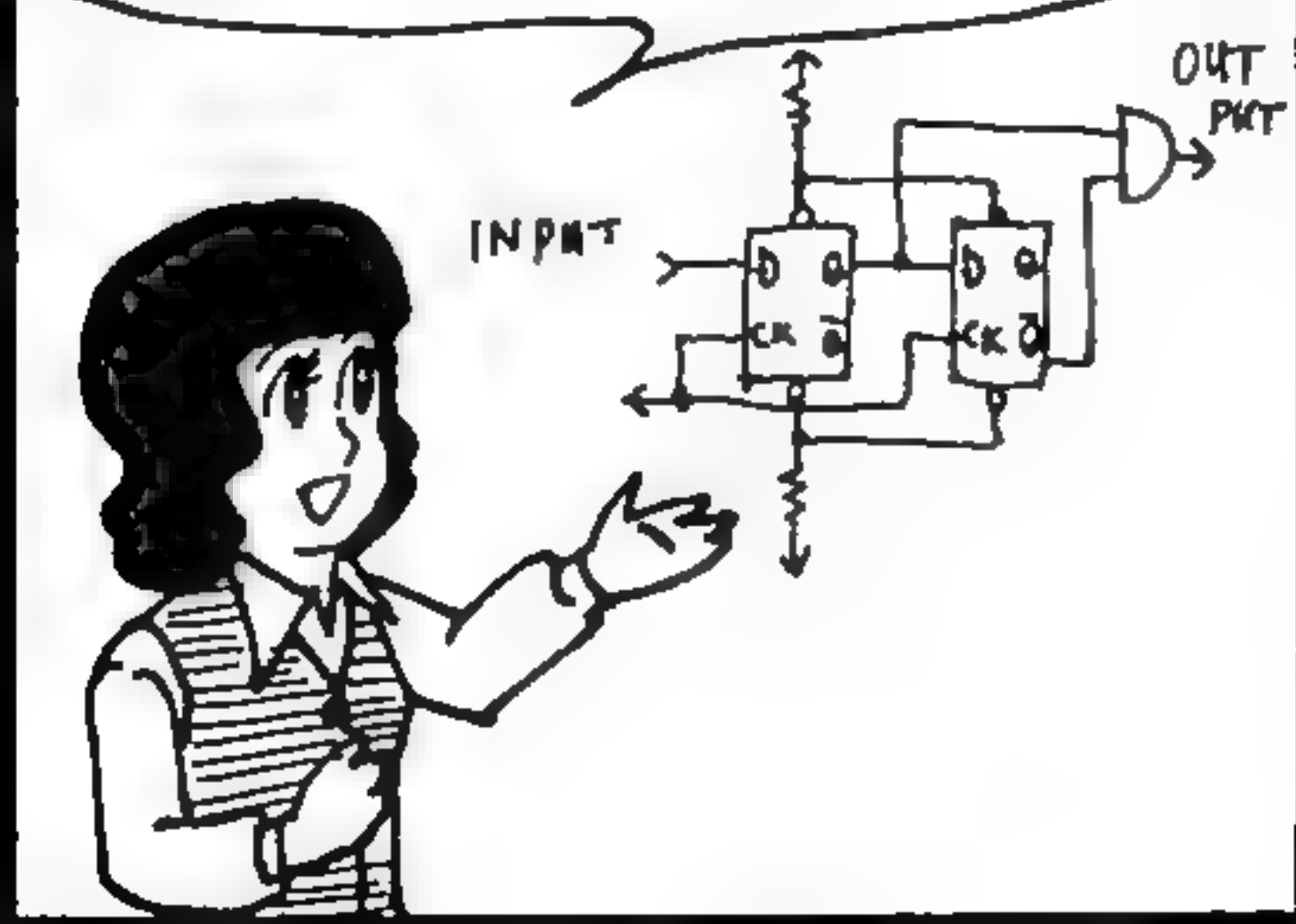
1発目のクロックで出力QがHにセットされ保持されます。2発目のクロックで入力がLだったら出力QはLに落ちます。



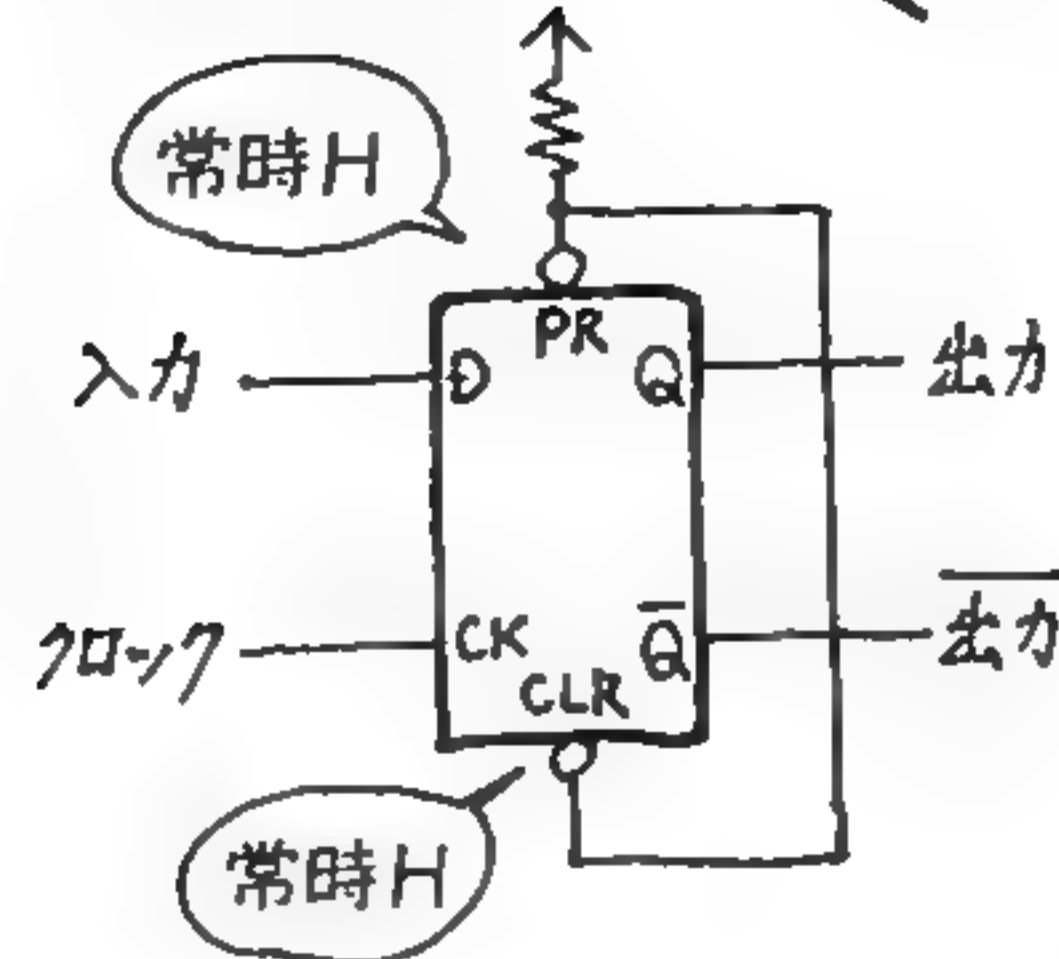
タイミングチャートを書いて回路が合っているか確かめてみましょう。



エッジトリガFF応用の基本です。よく、読み返して理解してください。覚えておけば回路図を読むのが楽になりますよ。

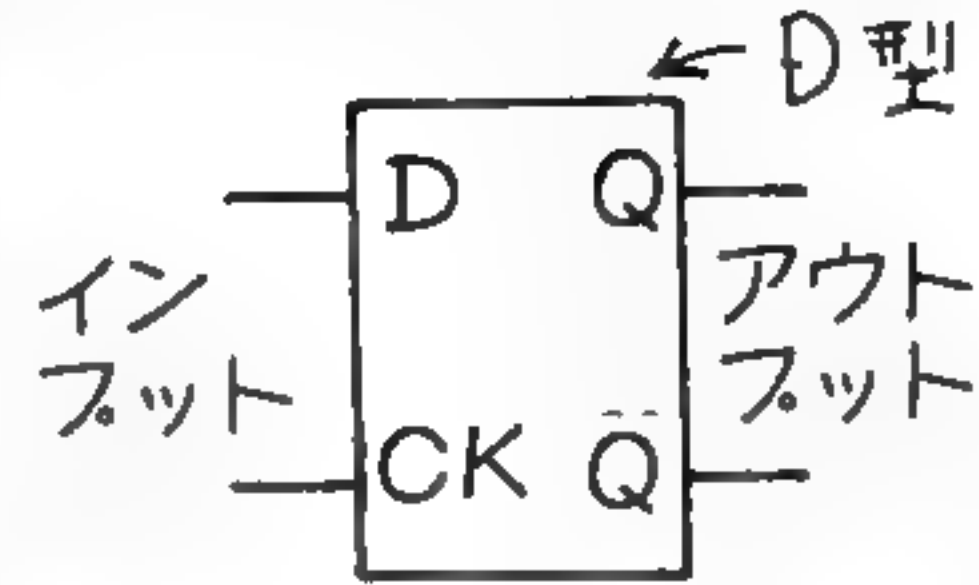


DタイプFFは回路図には、こんな形で通常書かれます。プリセットとリセットはプルアップ抵抗を介して常時Hにします。

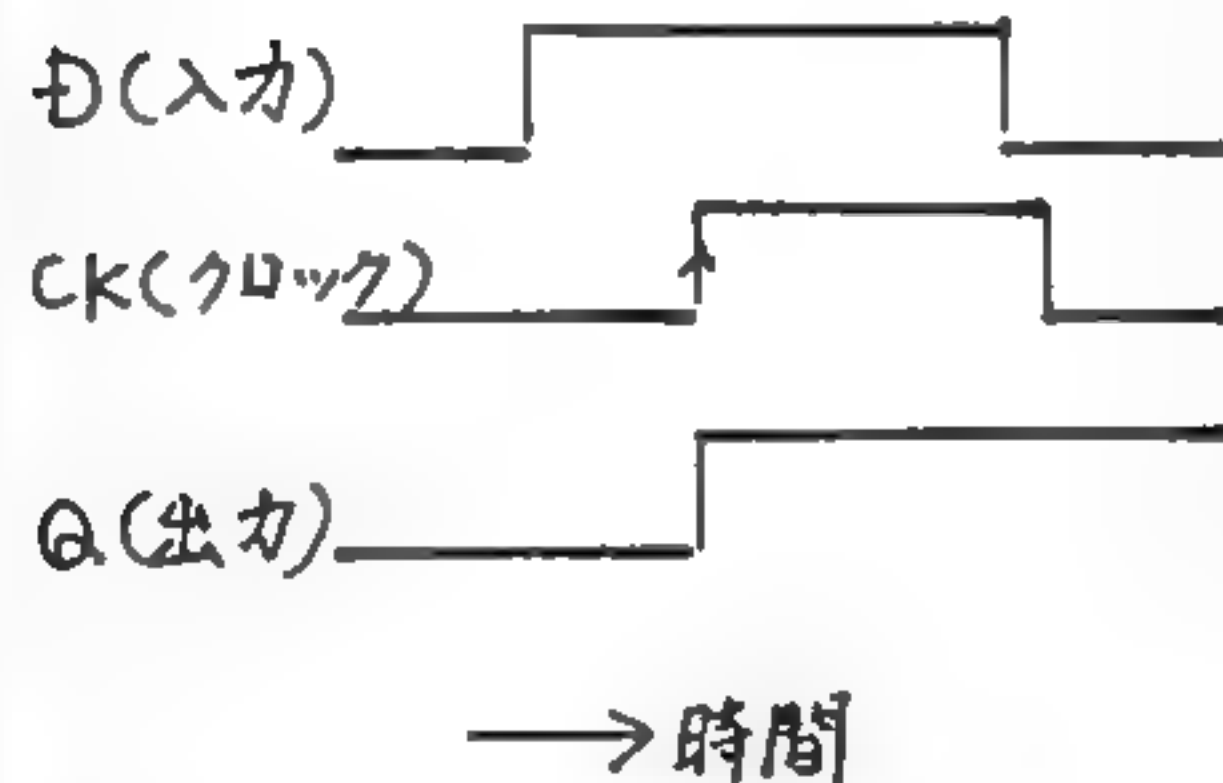


DタイプFFはこのような動作をします。

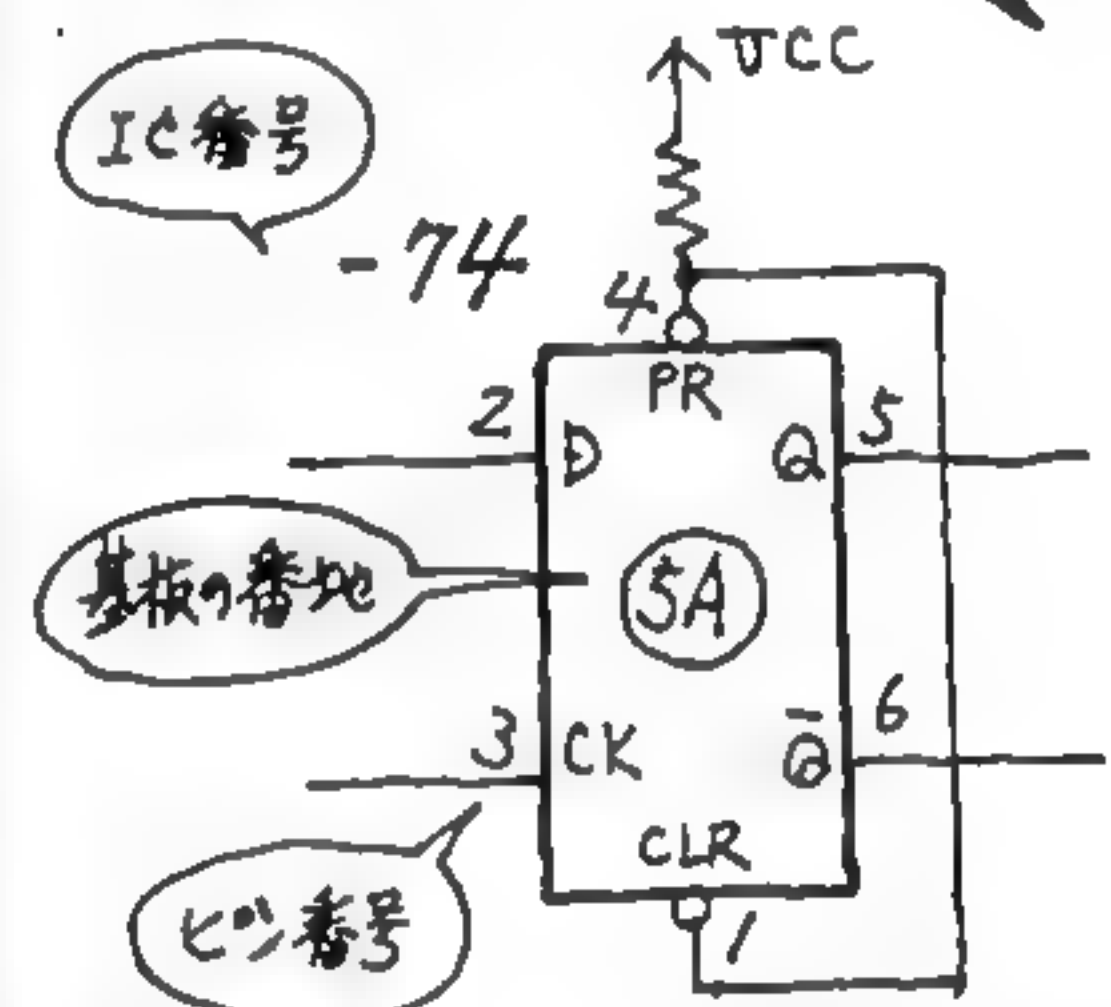
インプット		アウトプット	
CK	D	Q	$\bar{Q}$
↑	H	H	L
↑	L	L	H



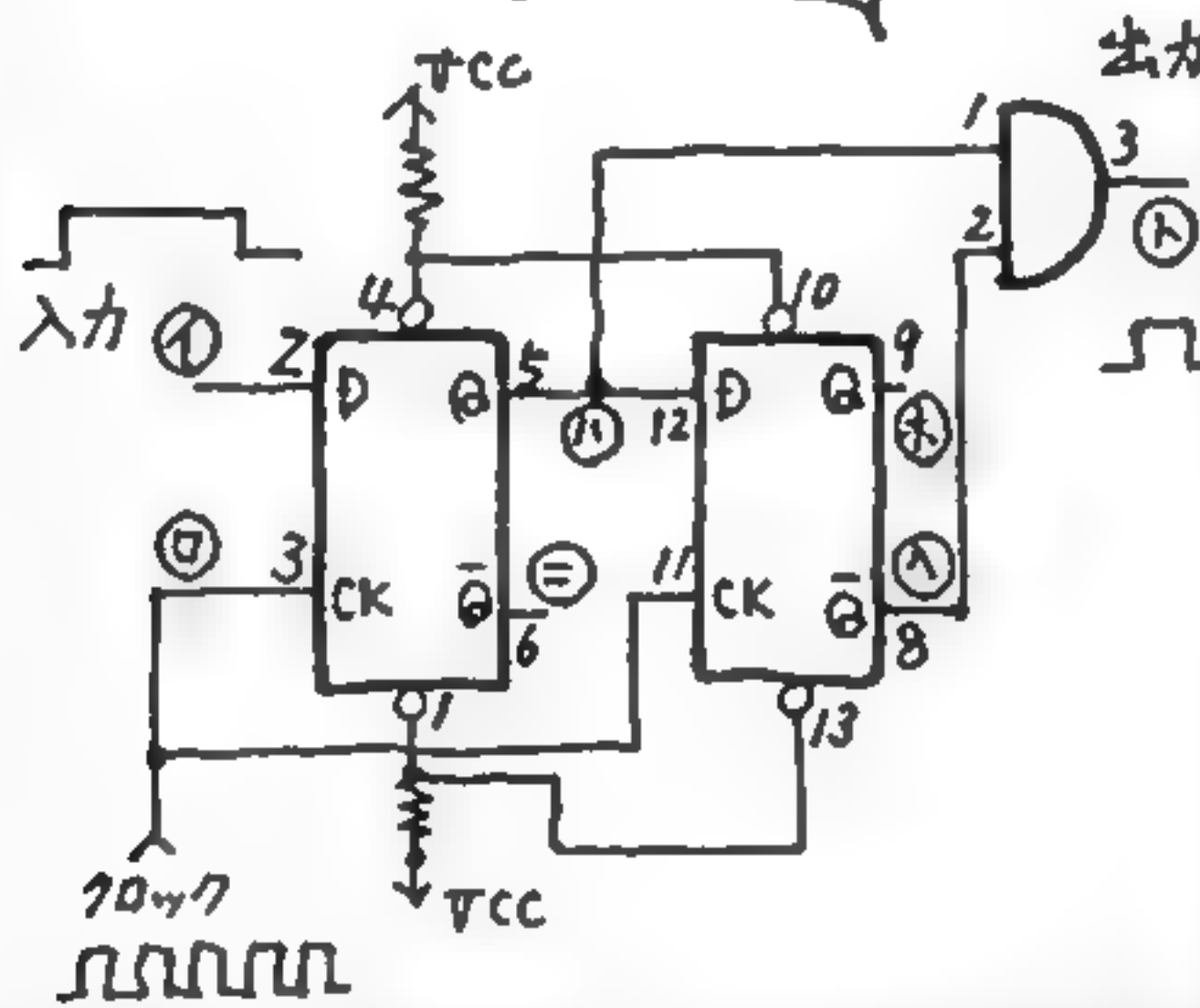
まず、正のエッジトリガート、入出力の関係を示すためにタイミングの例を示します。



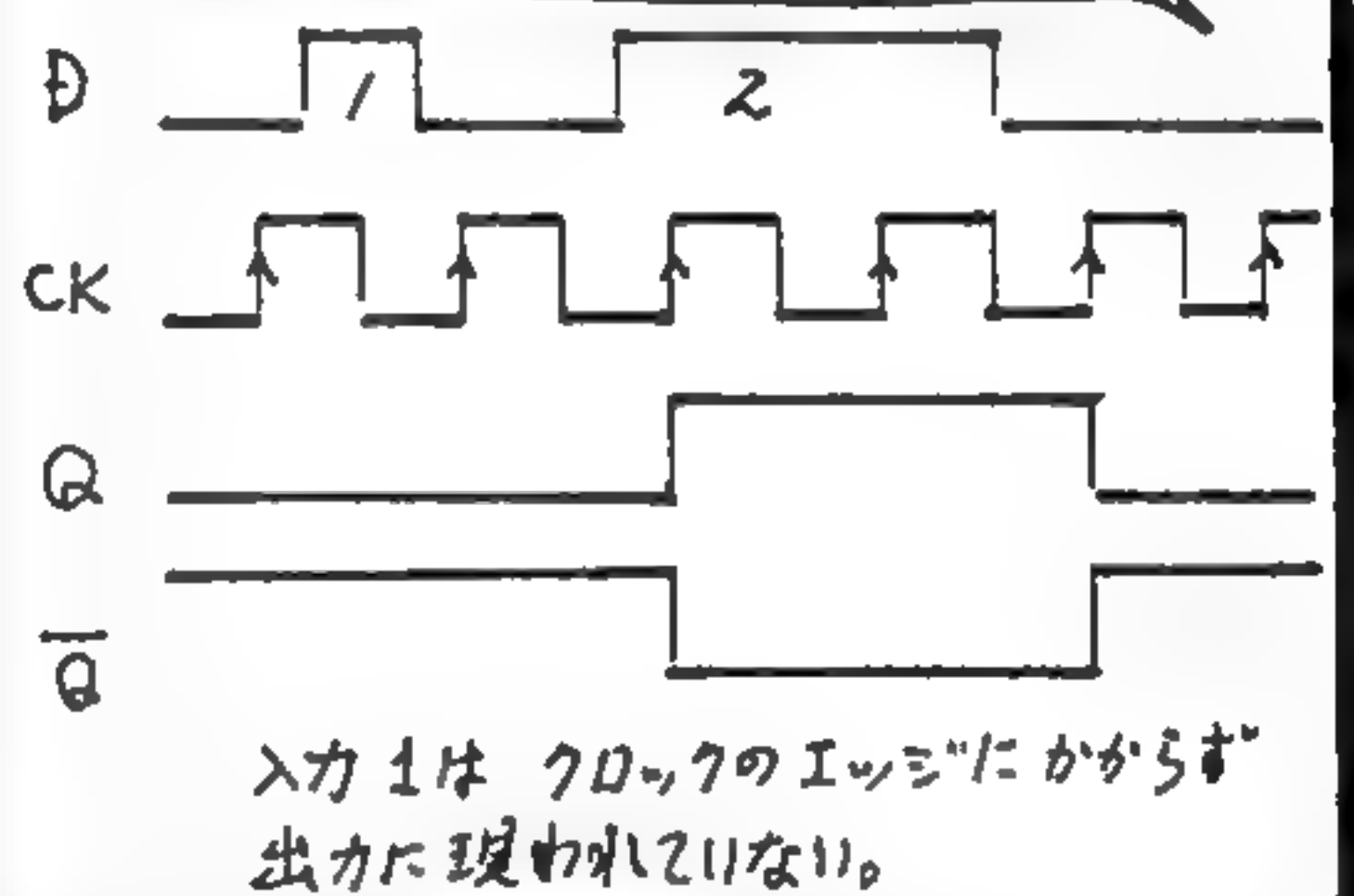
図面として書く時は、パッケージNo.、記号、ピンNo.、基板の番地を書いておきます。



この性質を利用してクロック1周期分のパルスをつくることができます。これがその回路です。

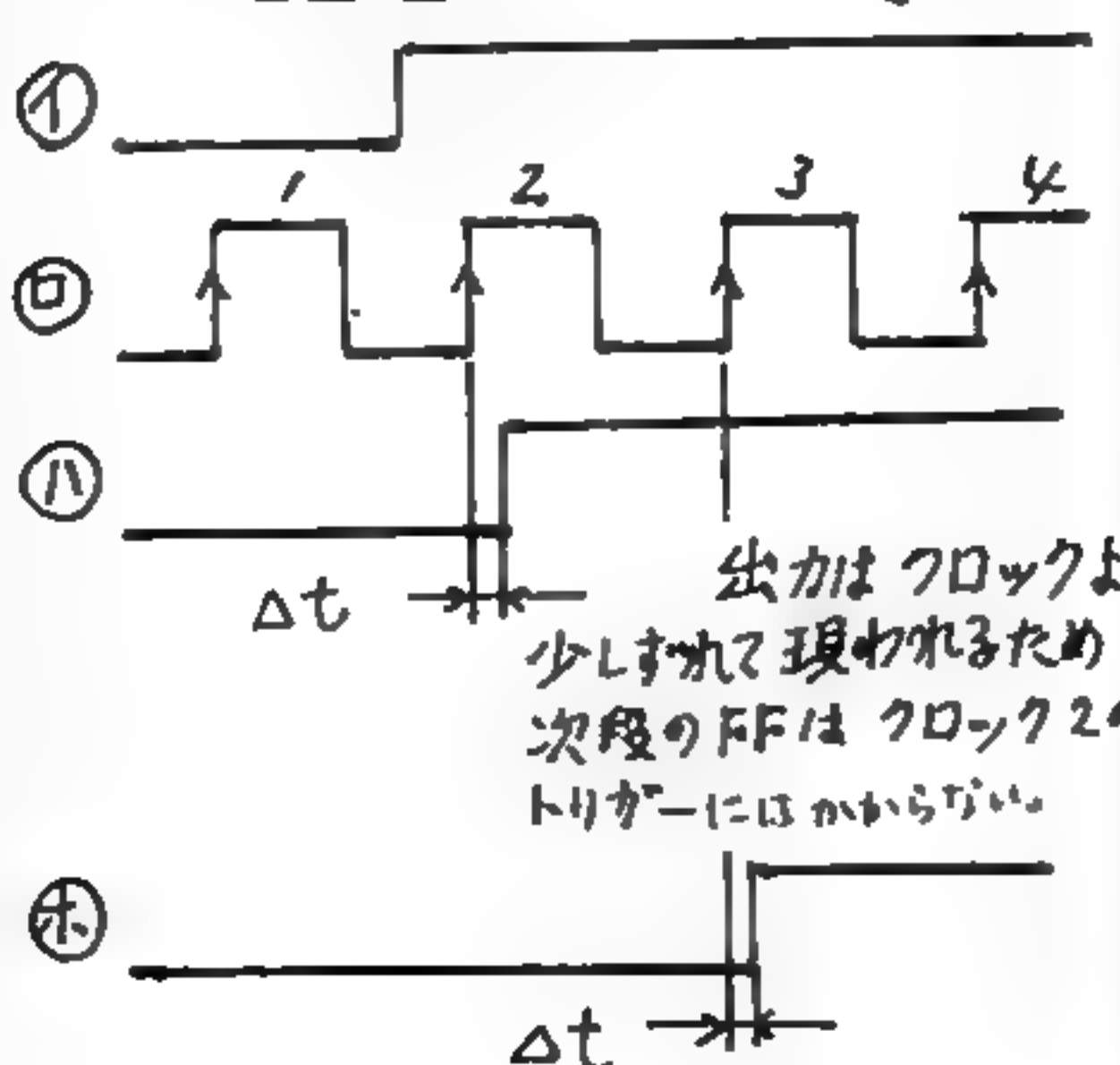


このタイミングチャートを見てもわかるようにクロックのエッジにかからない入力は無視されることになり、ノイズに対しても強い回路が設計できます。

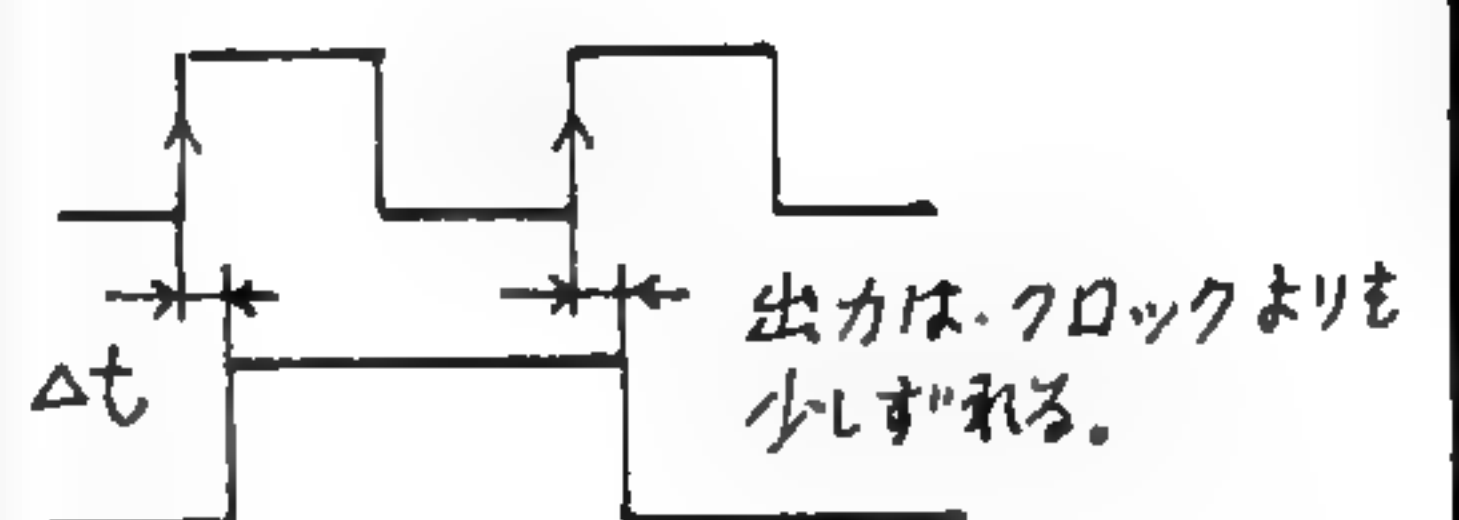


入力1は、クロックのエッジにかからず出力に現われていない。

これは、エッジトリガだから、直列につなぐと1クロック分ずれることを応用したものです。



前のコマは、入力パルスの最初の方で1クロック分のパルスが得られることを示しています。



出力は、クロックよりも少しずれる。

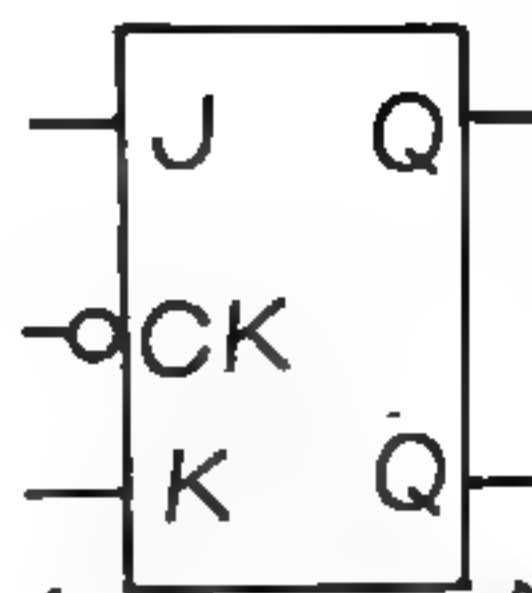


クロック	インプット		アウトプット	
CK	J	K	Q	$\bar{Q}$
	L	L	Q0	Q0
	H	L	H	L
	L	H	L	H
	H	H	TOGGLE	

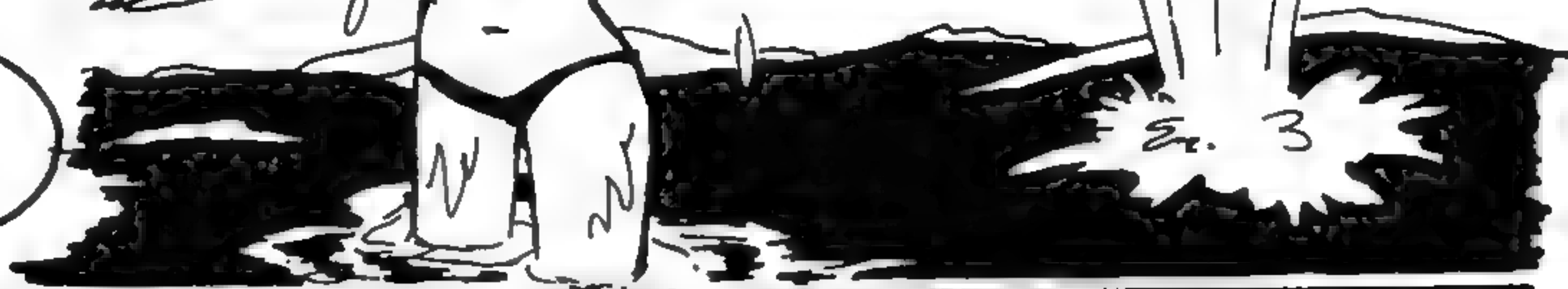
クロックが  
↓となる前の  
状態を保持し  
ます。

JK型はセット  
入力とリセット  
入力を持ったFF  
です。

JK型  
FF



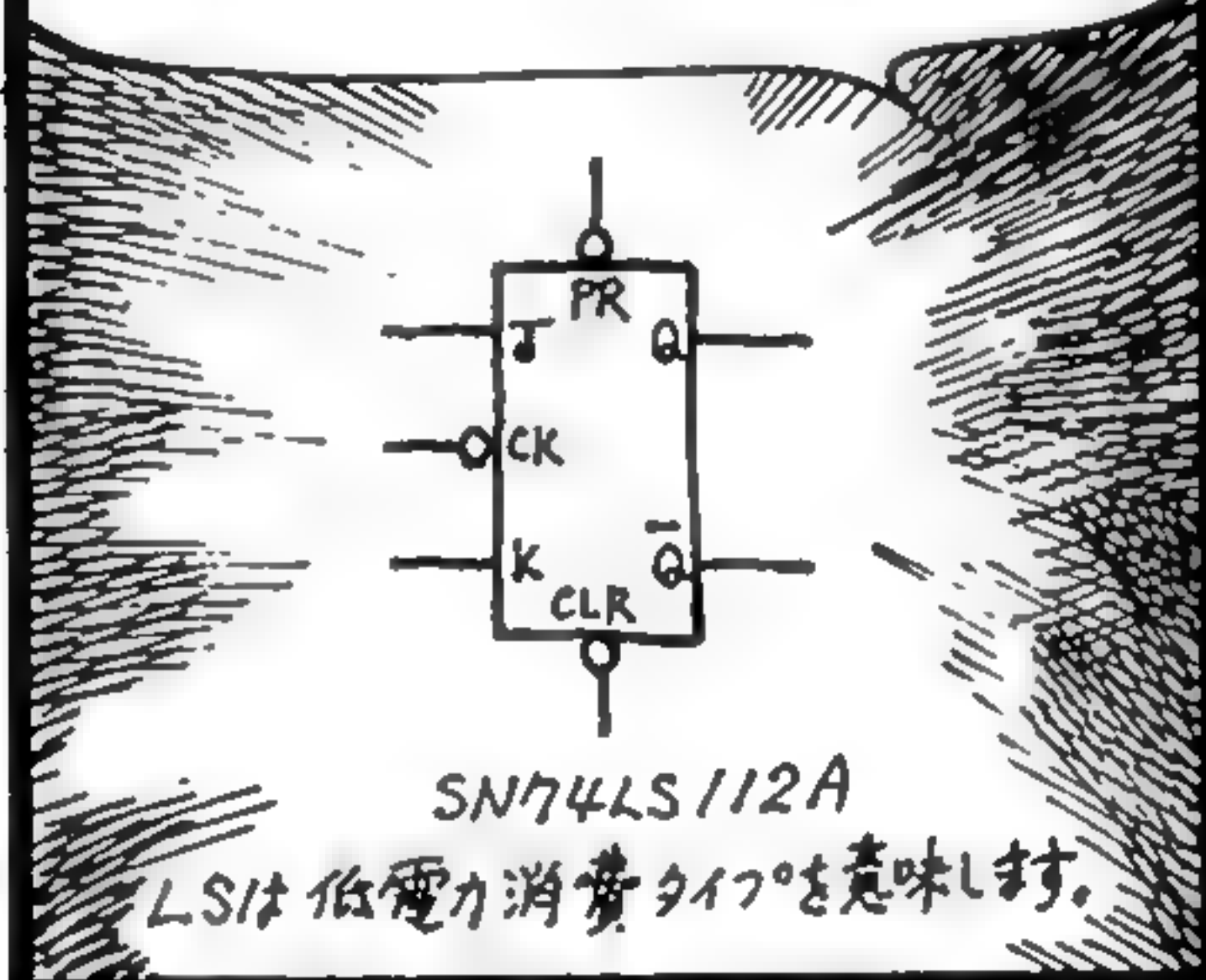
クロックが↓となる前の  
状態から反対の状態に  
変わります。



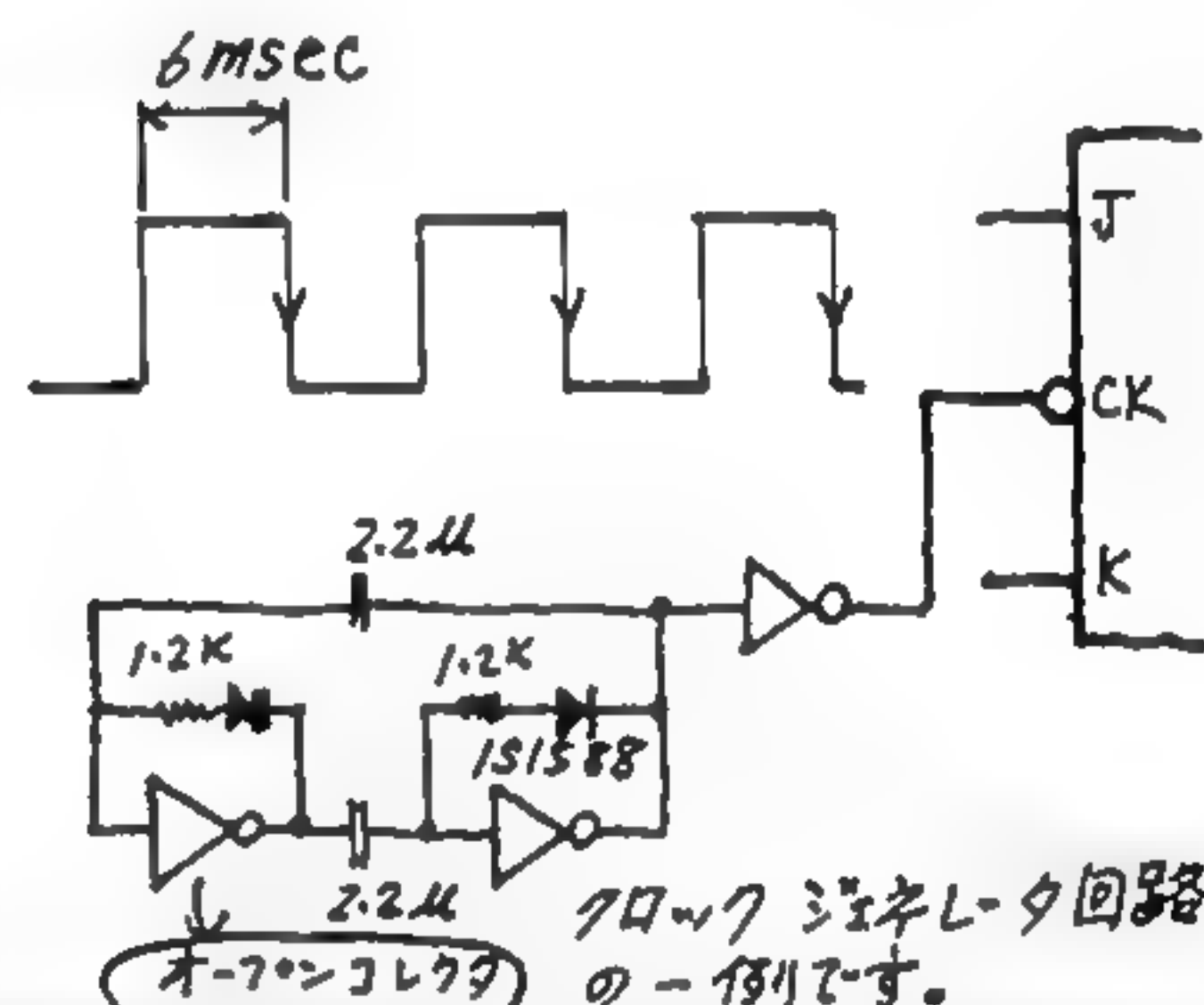
‘セットする’とは出力QをH  
にすることだと考えてくださ  
い。QはQの反対ですからL  
になります。



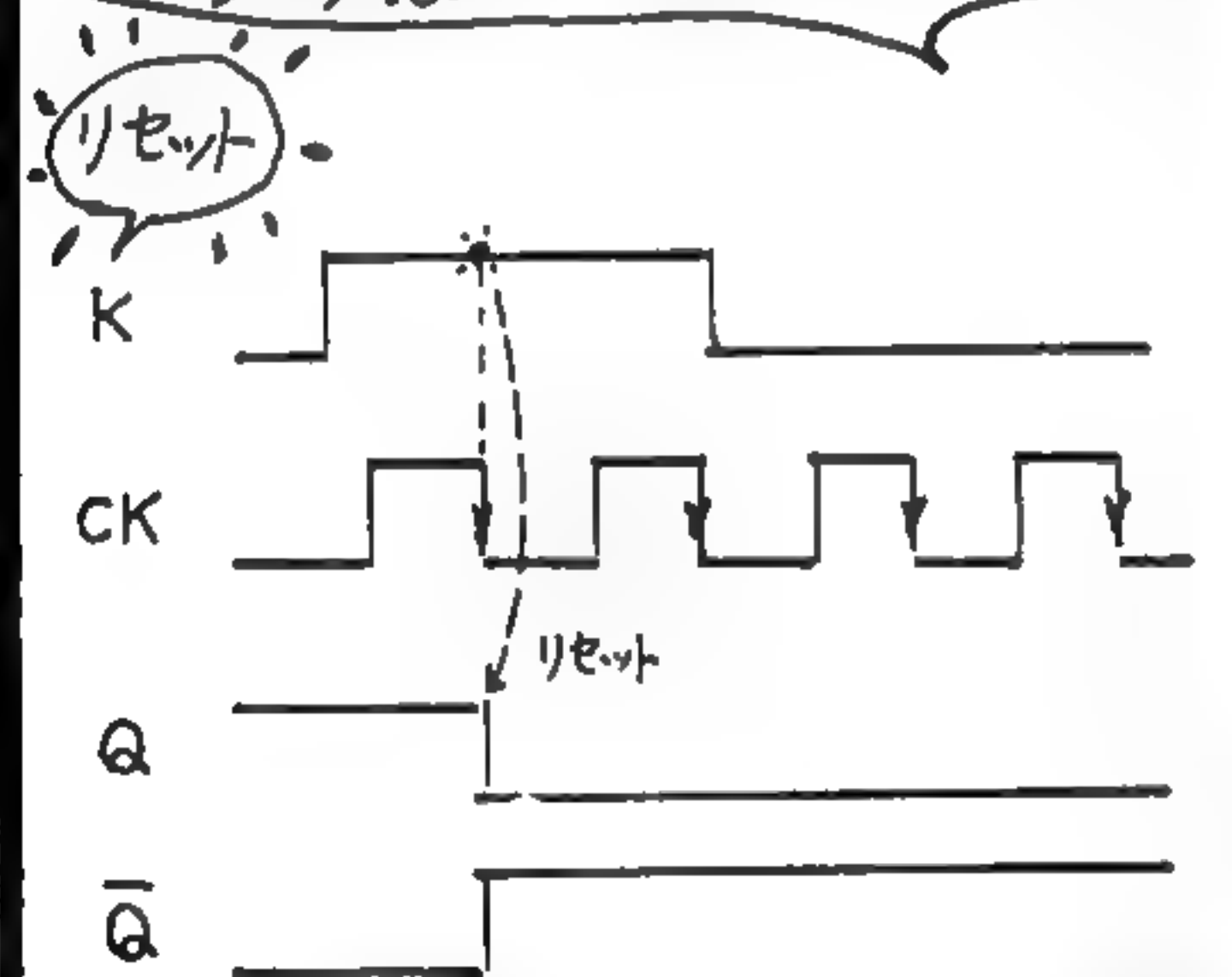
JKにもいろんな種類 (トリガー  
が正エッジのものもある。)の  
ものがありますが、ここではSN74LS  
112Aを例にとってみます。



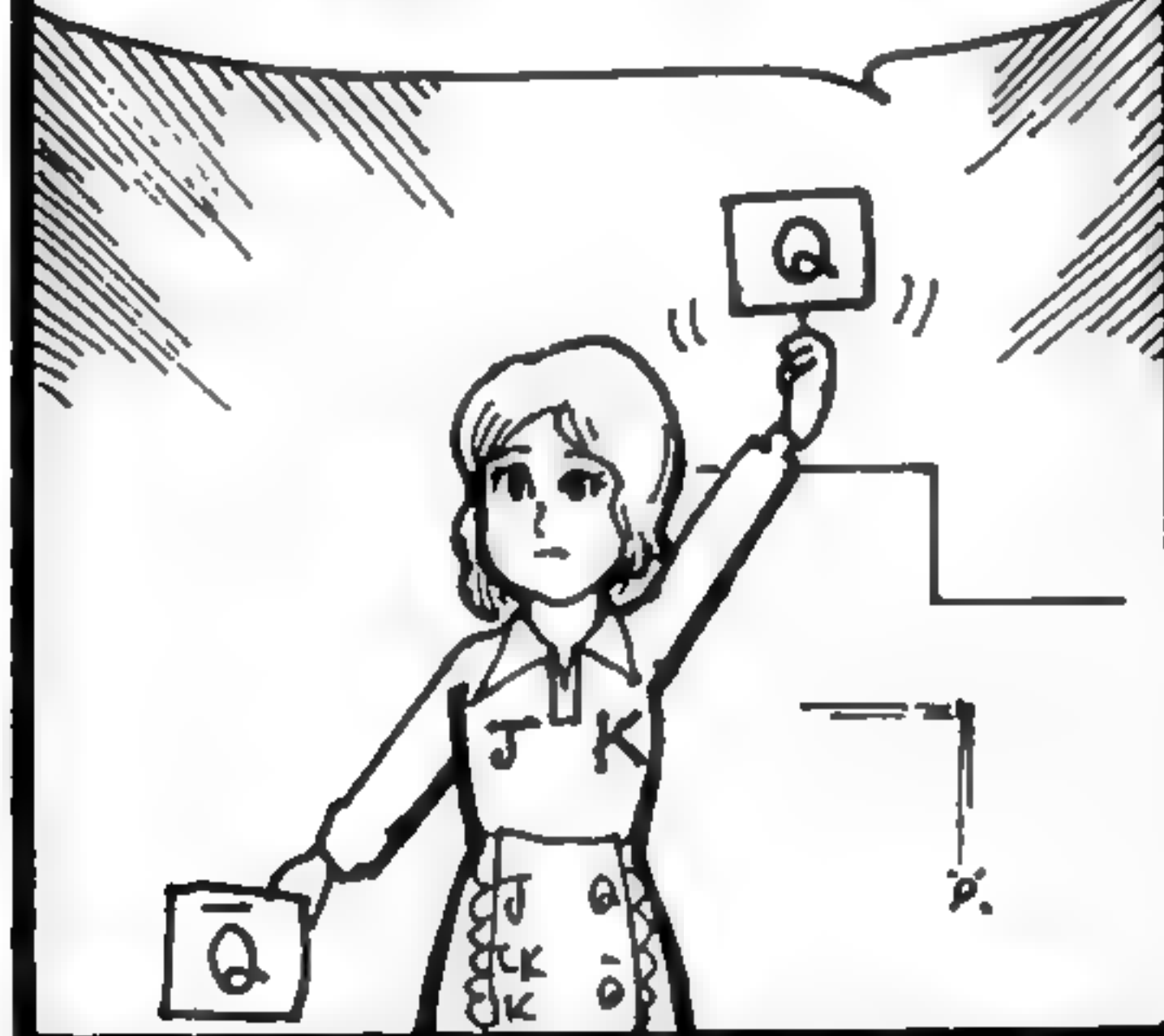
出力を変化させるタイミン  
クをとるクロックのエッジは負  
側です。



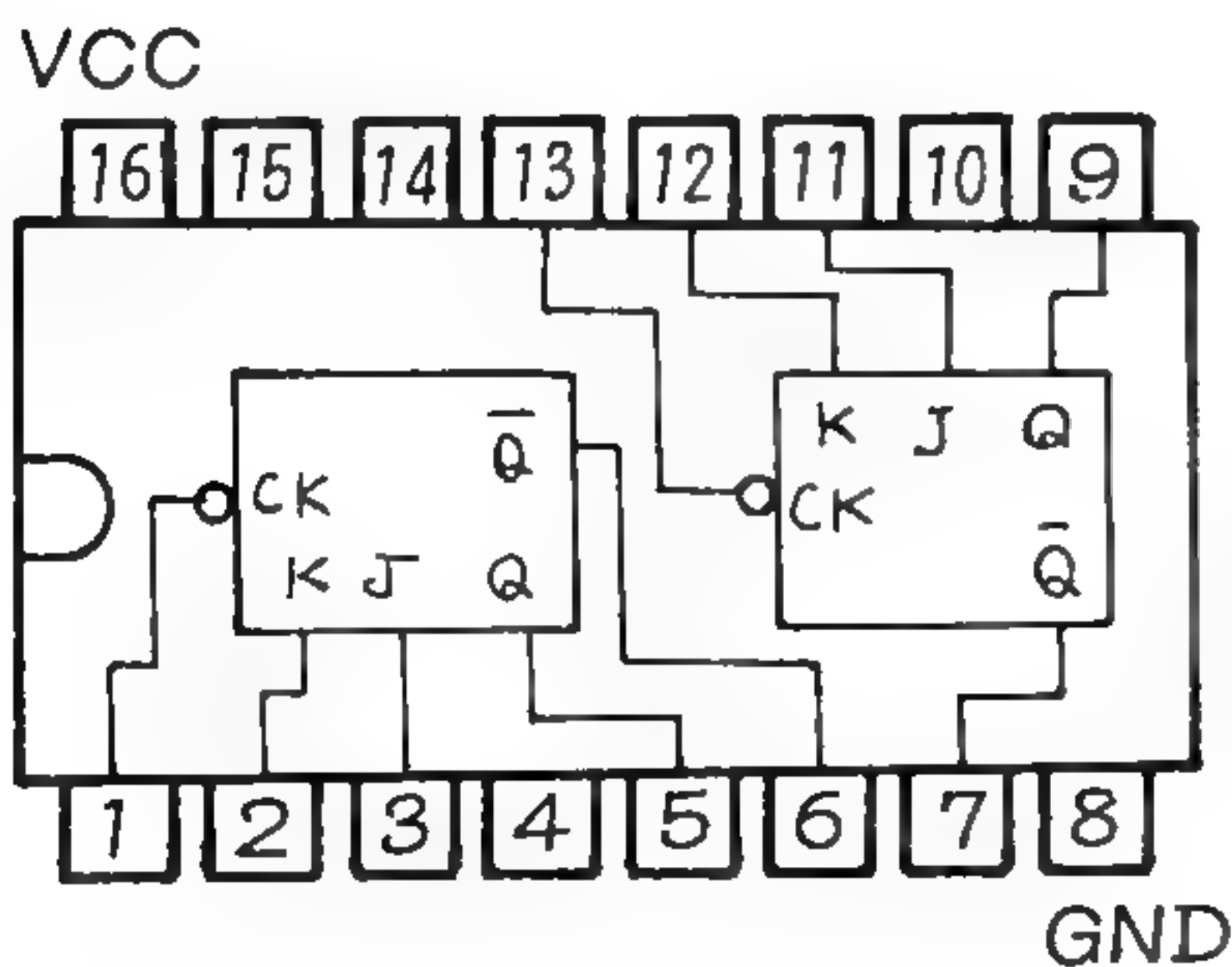
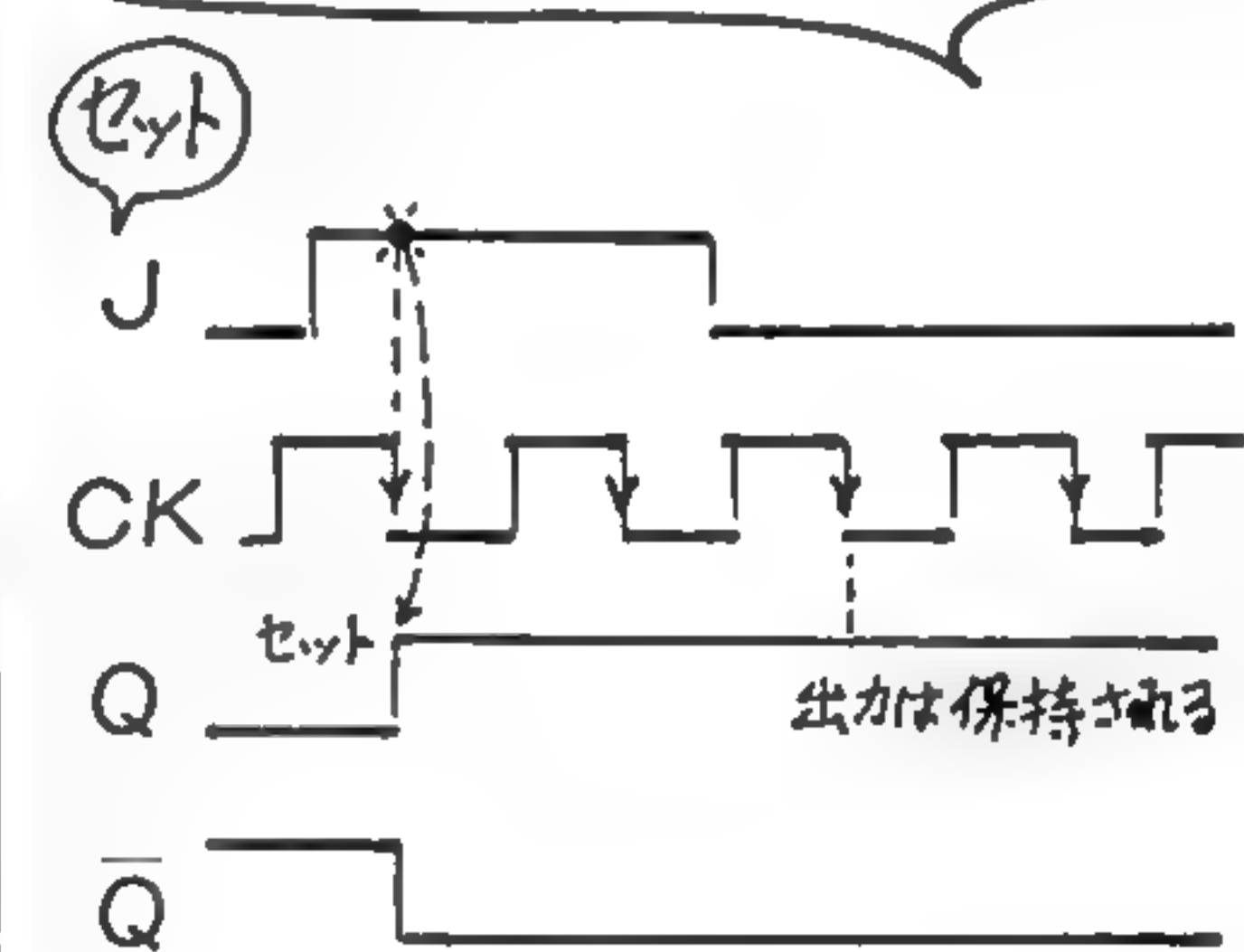
そして、出力Qがリセット入力K  
によって、Lに落ちるタイミン  
グは……



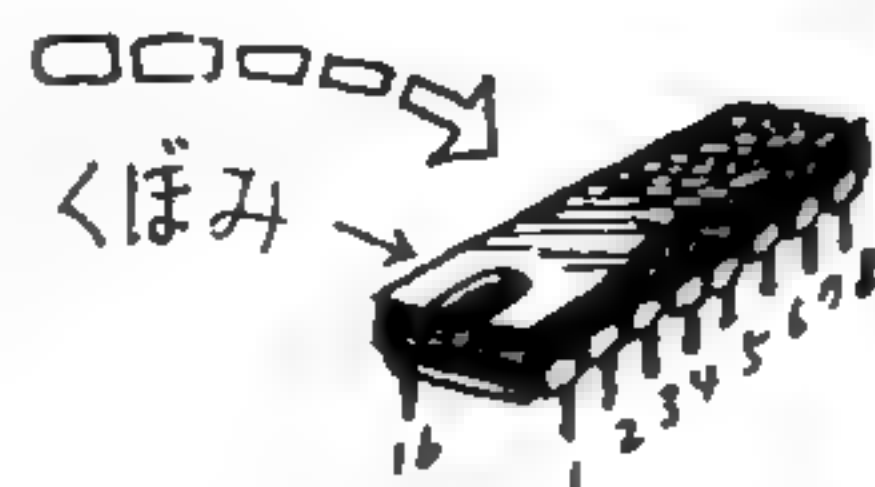
入力パルスがLに落ちてから  
も出力QはHに保持され続け  
ます。



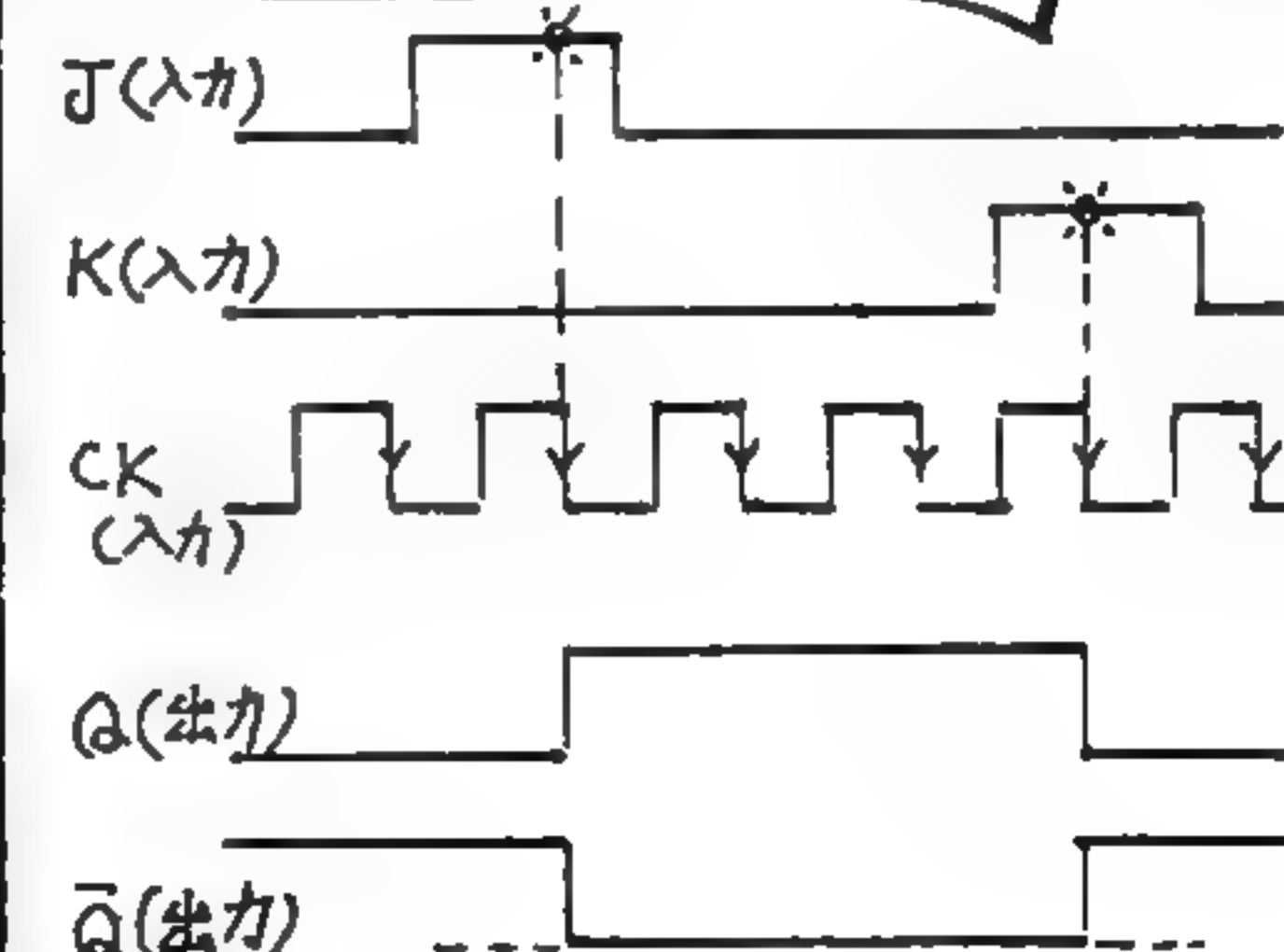
まずQが入力Jによってセッ  
トされるタイミングを見てみ  
ましょう。



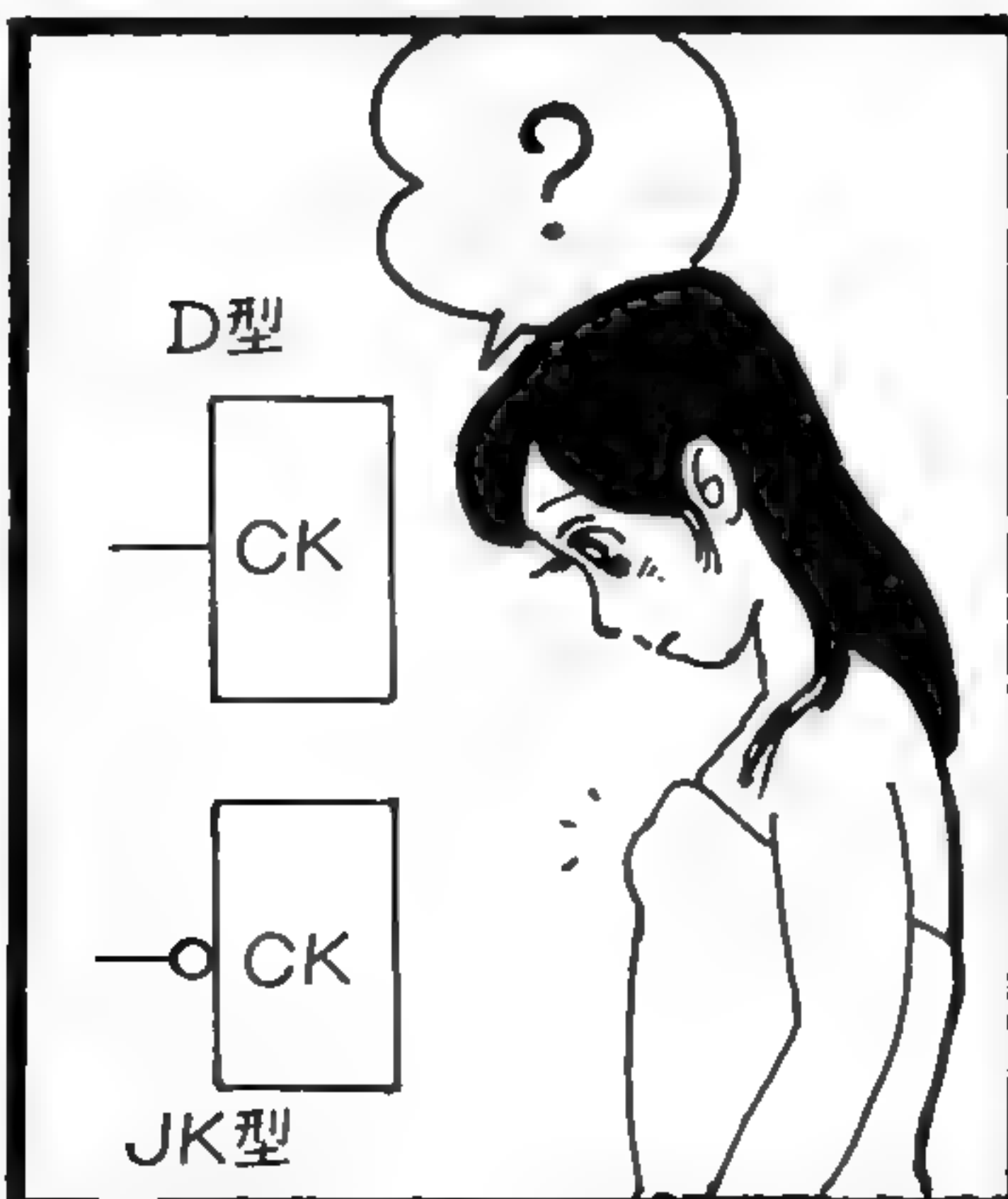
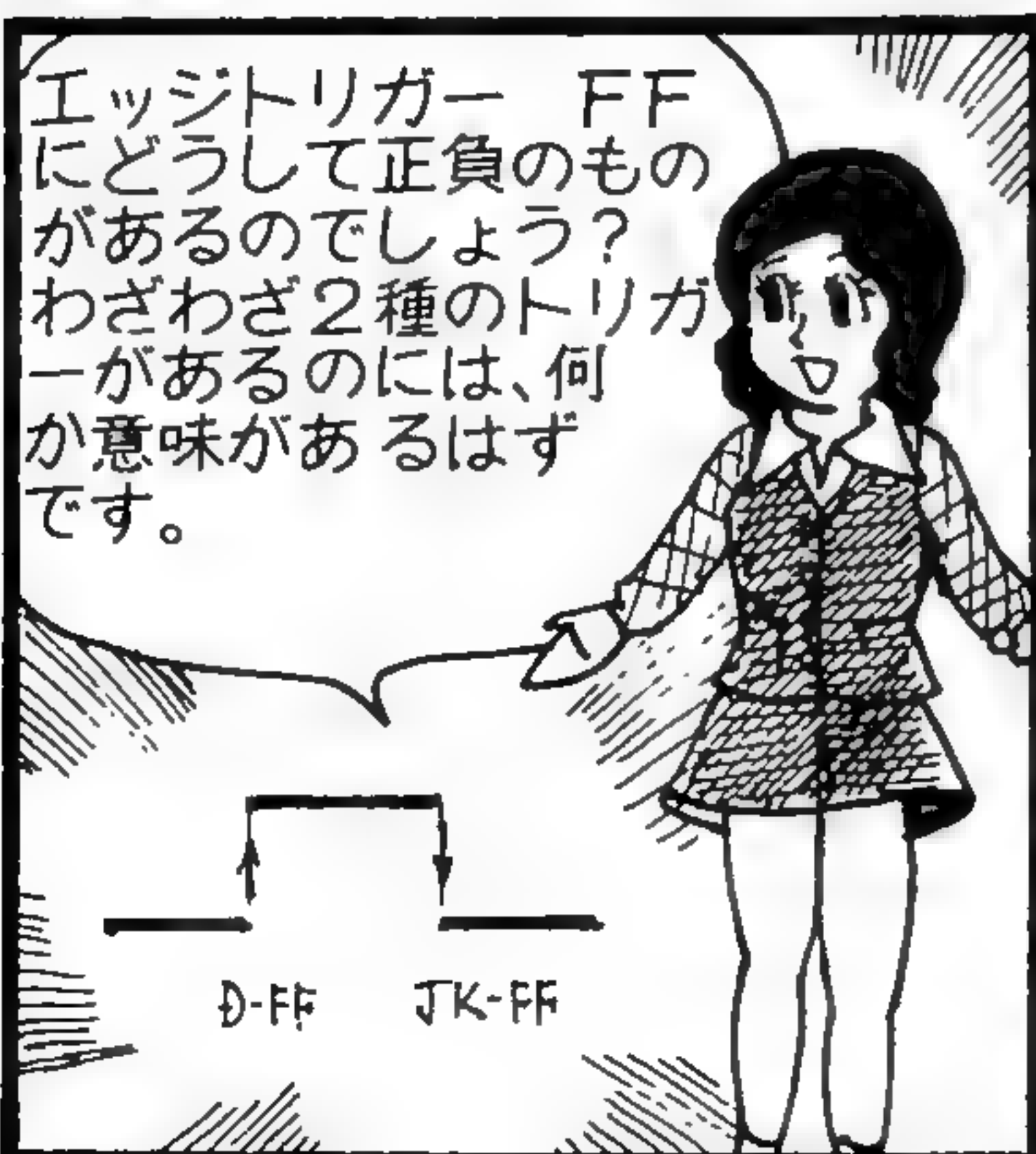
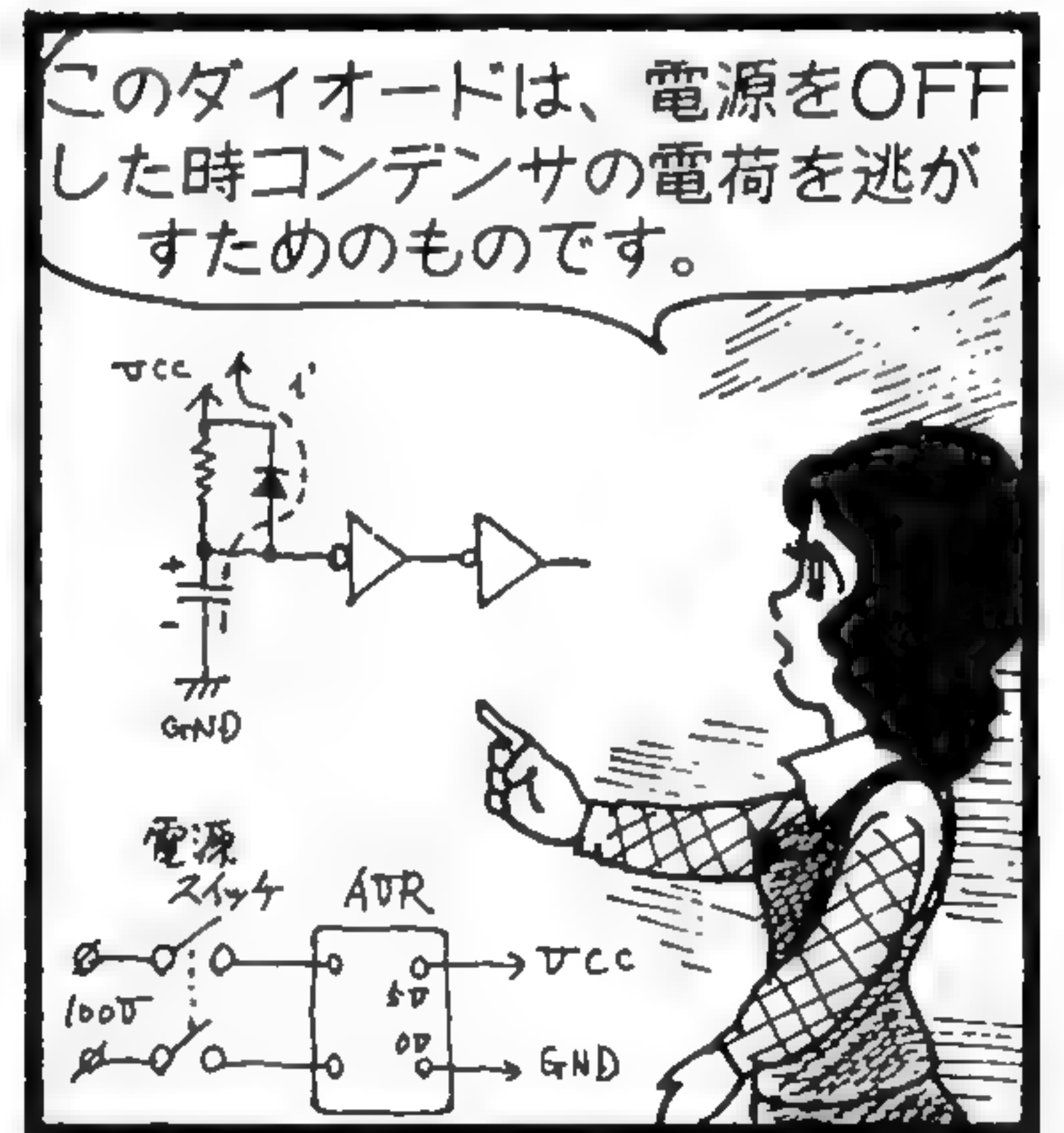
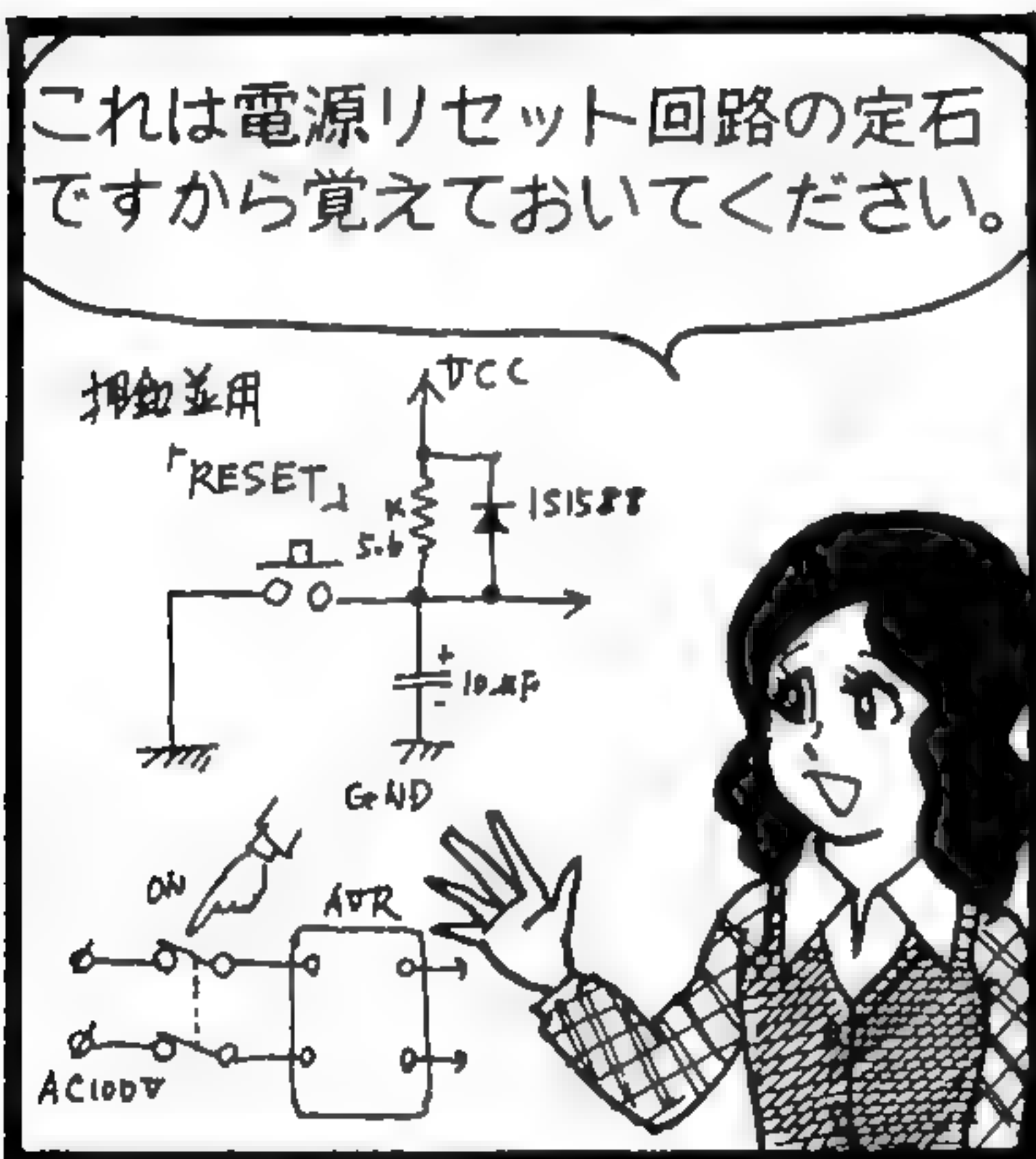
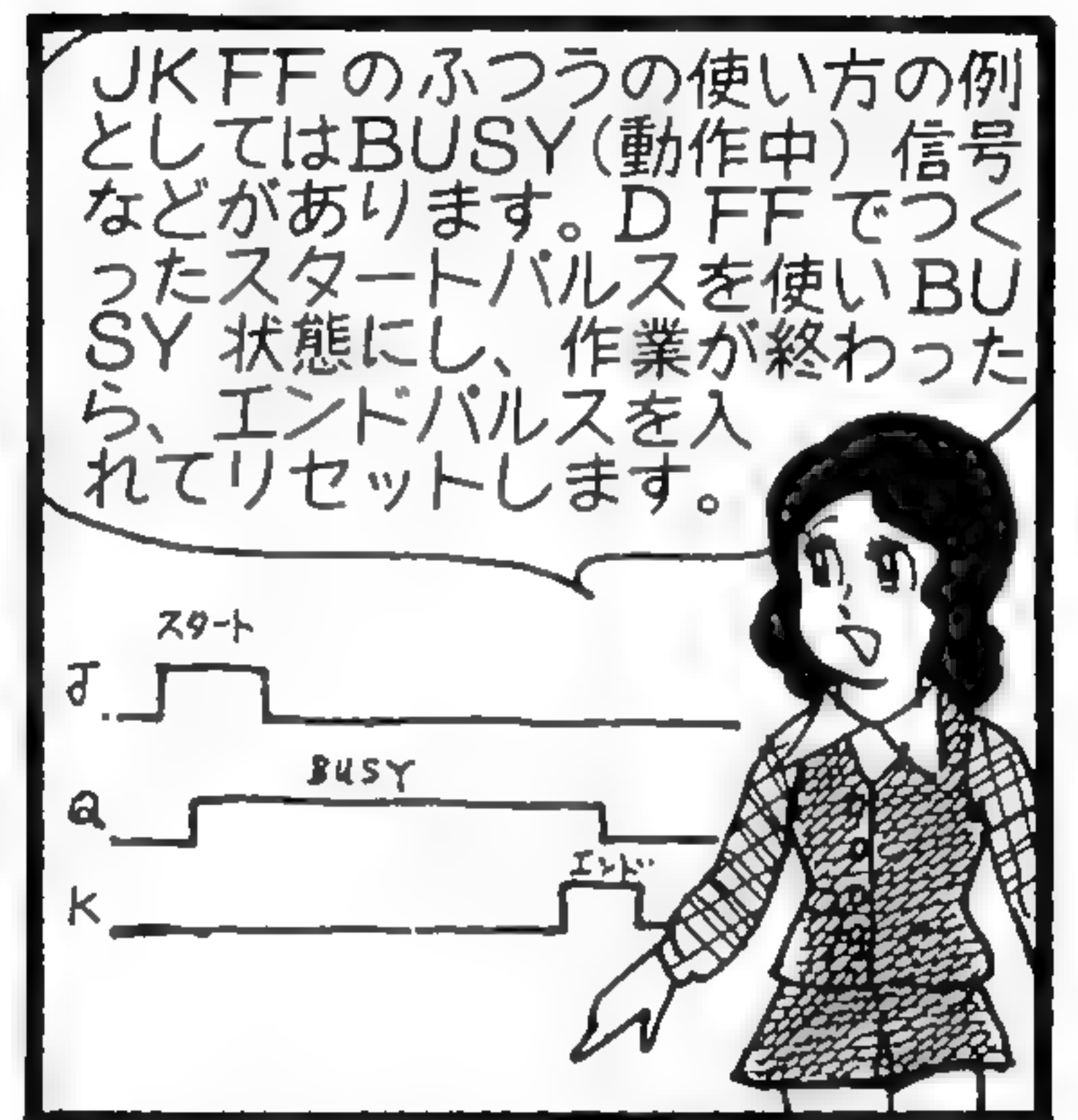
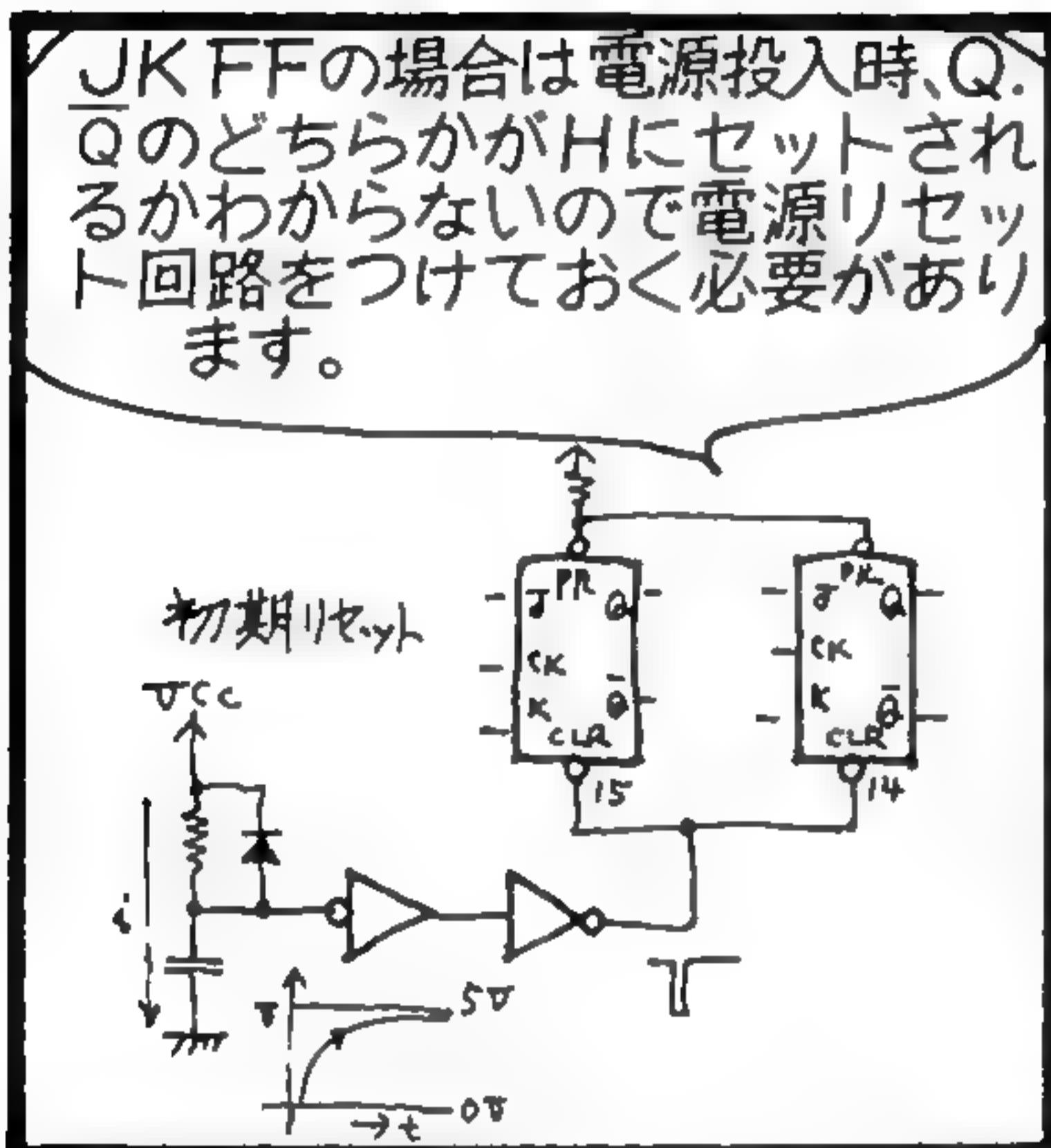
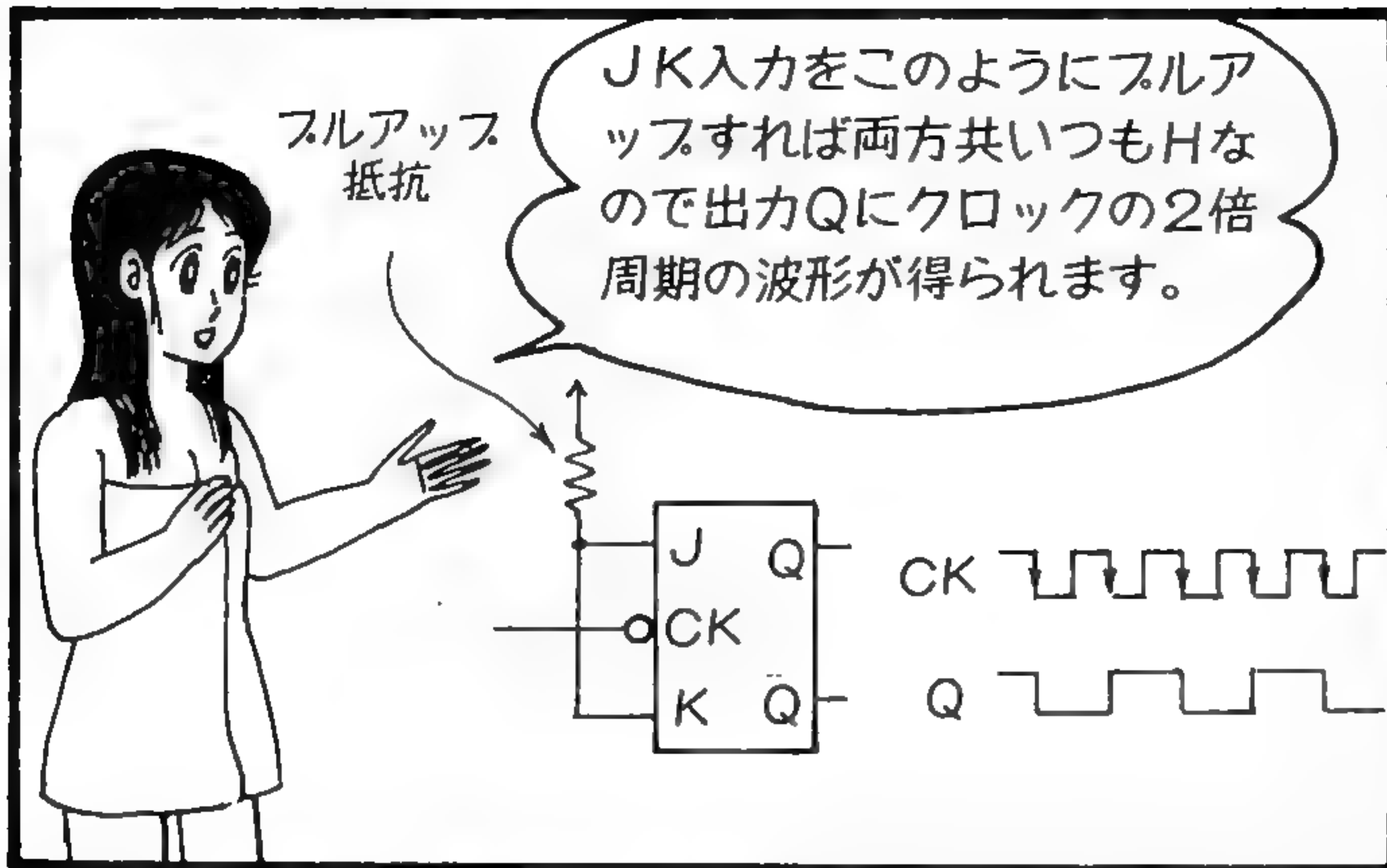
JKもふつう2個がひ  
とつのパッケージに入  
っているものが売られ  
ています。SN74LS11  
2Aは16ピンです。SN  
7474は14ピンでした。



このセット・リセットをひと  
つにまとめるとこのようにな  
ります。



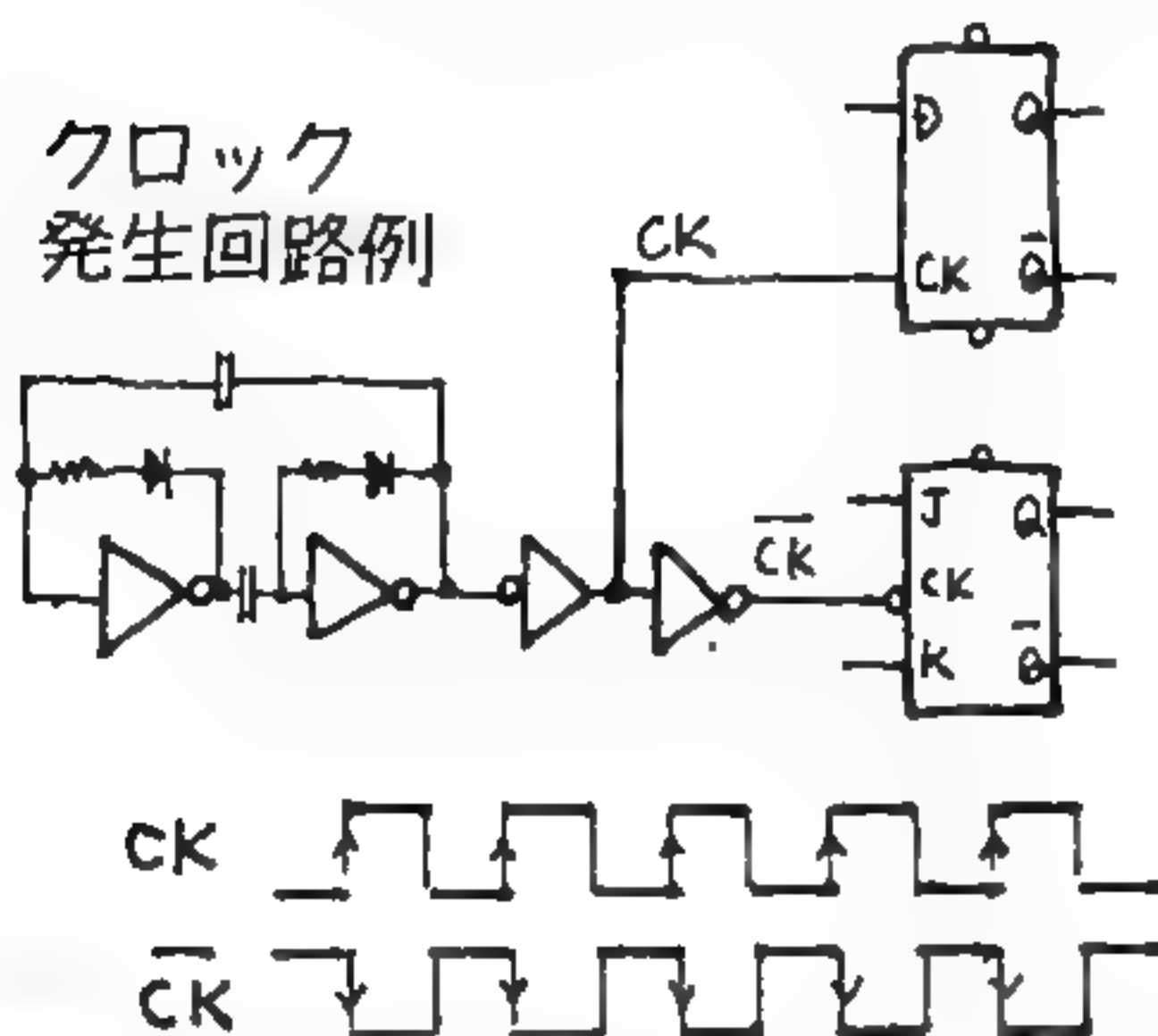




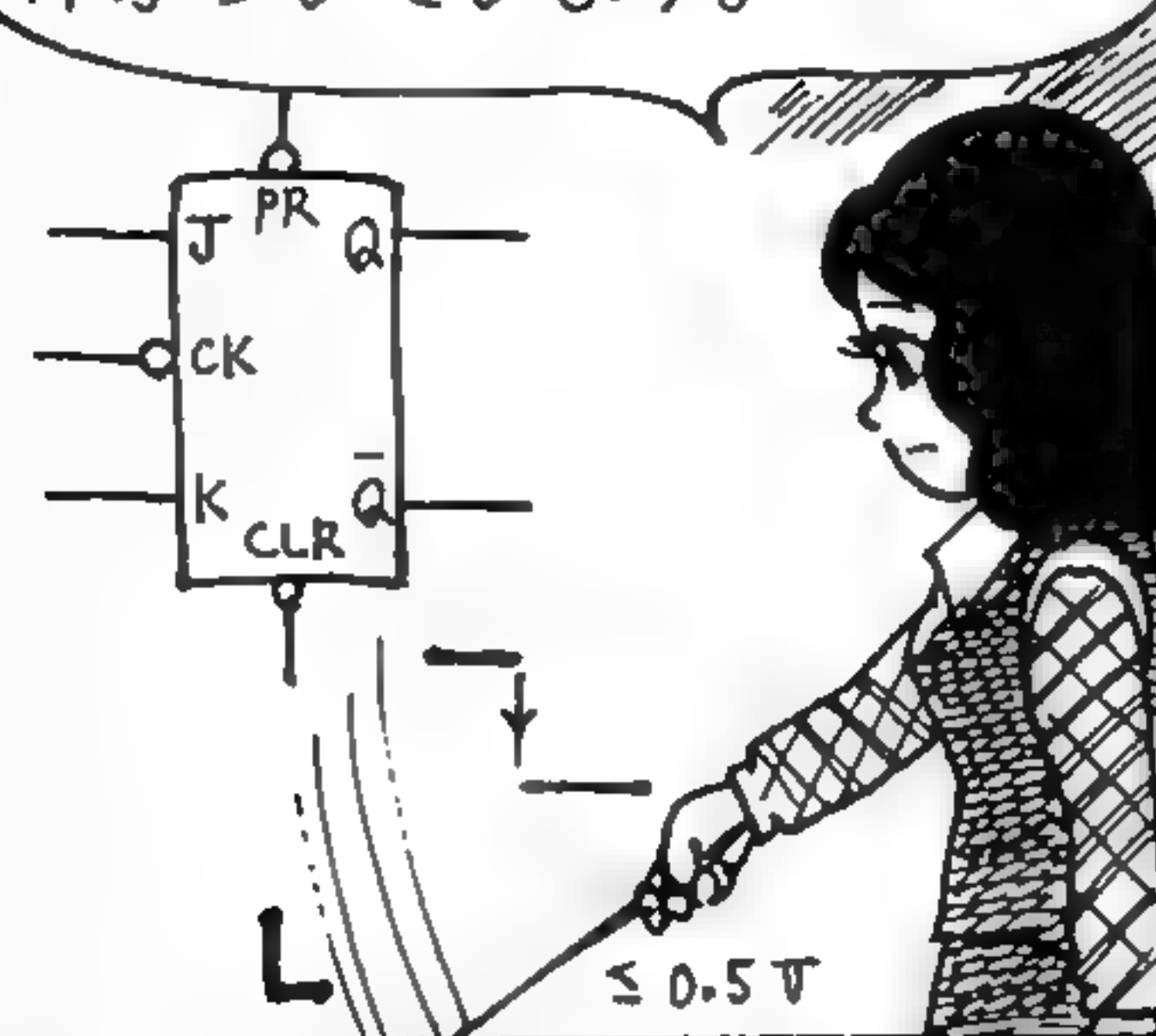


これは単純に考えると、DとJK FFのクロックはCKとCKの関係がある' と思ってしまいかもしれません。

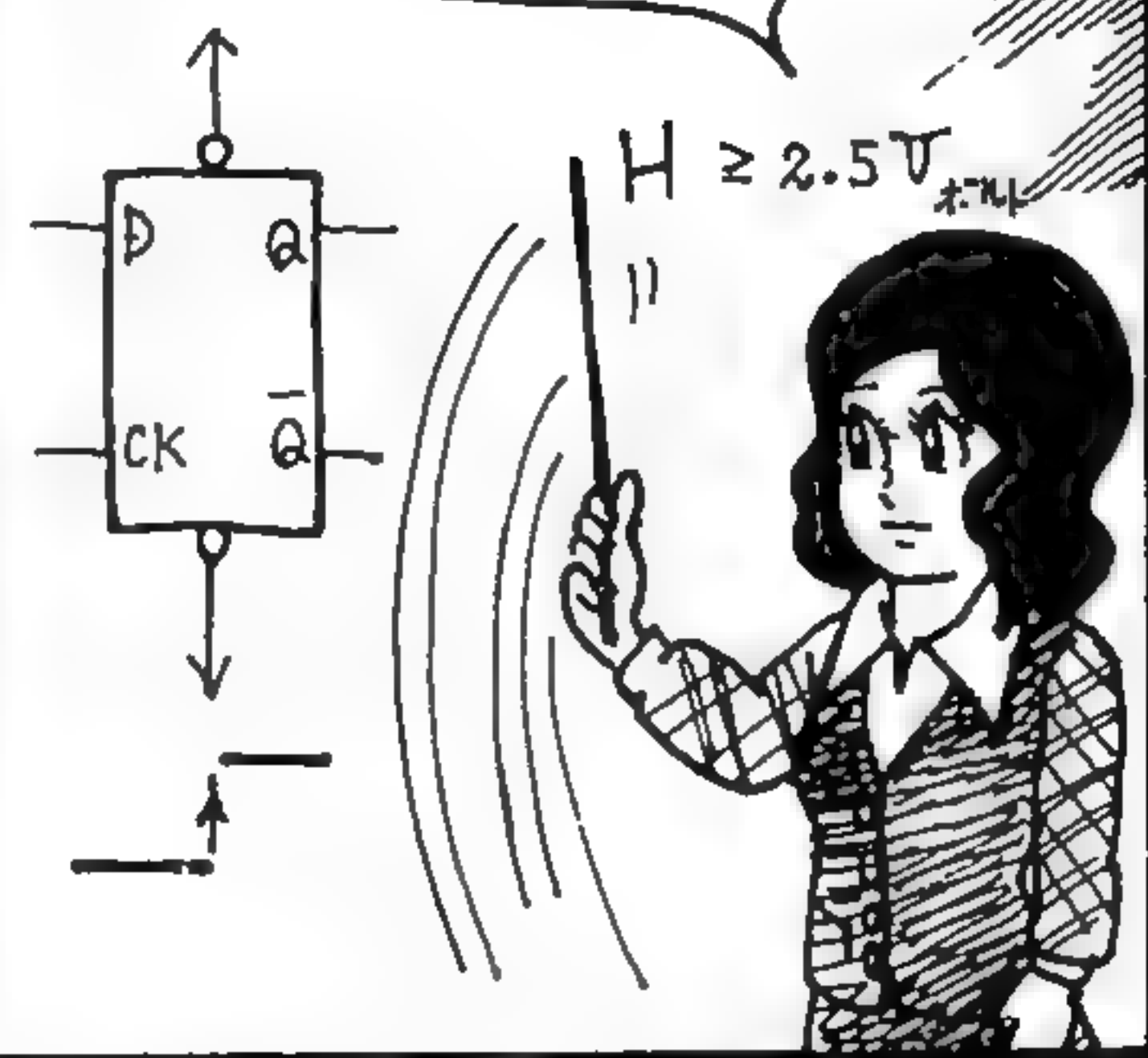
クロック発生回路例



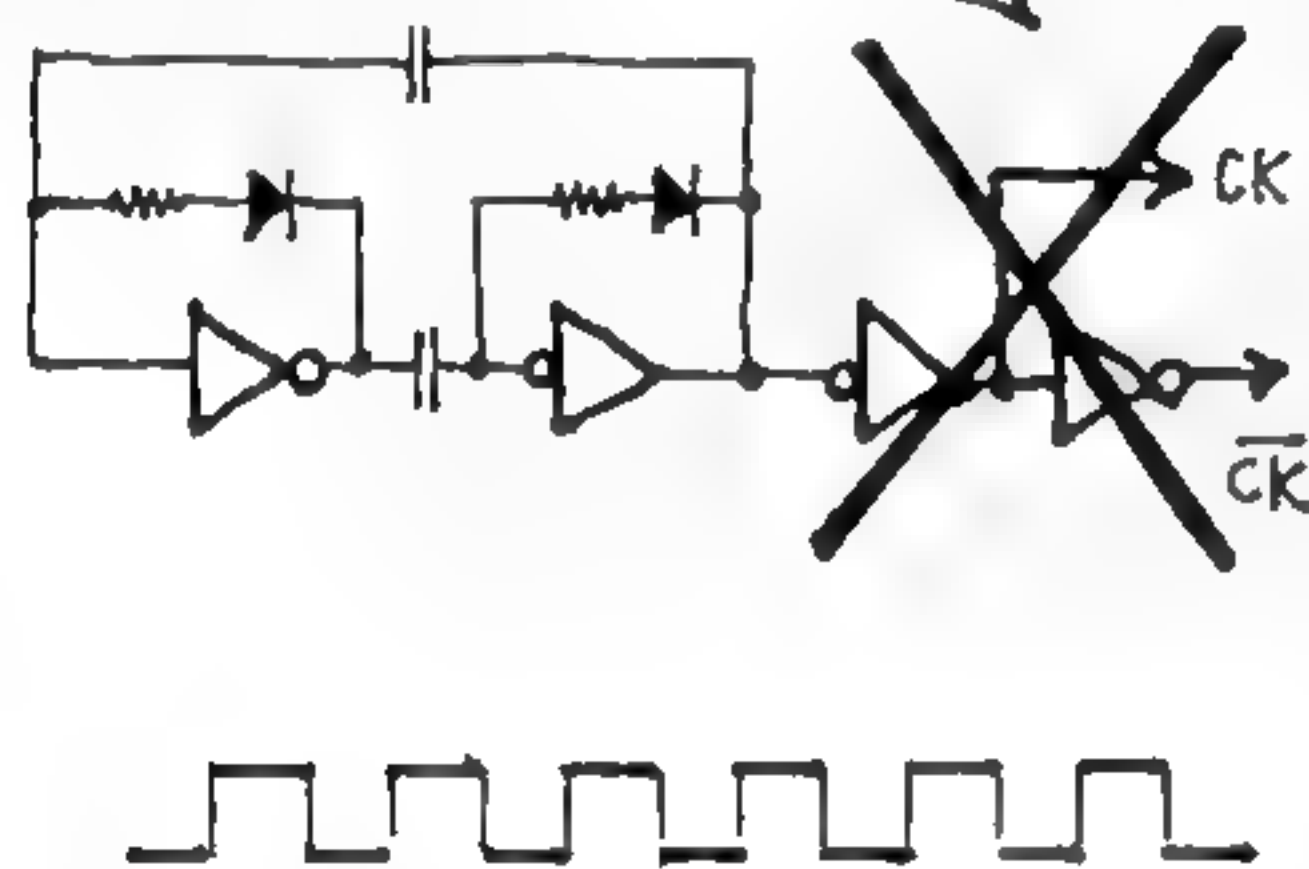
JK FFはクロックの立下りでトリガーがかかりました。こっちのCKには例の小さな○印がついています。



D FFはクロックの立上りでトリガーがかかり、



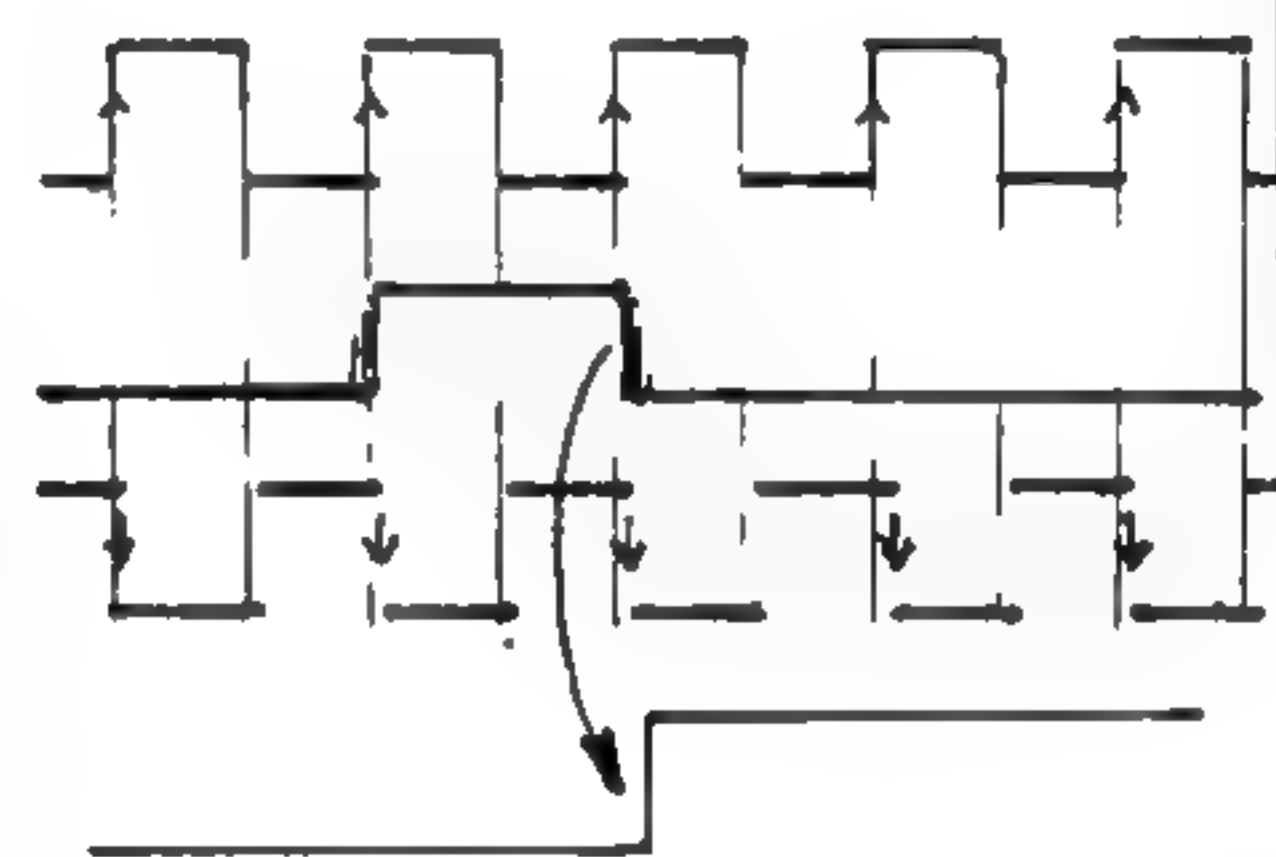
クロックをCKとCKにするのは考えたようで考えてないのです。単なる早トチリです。



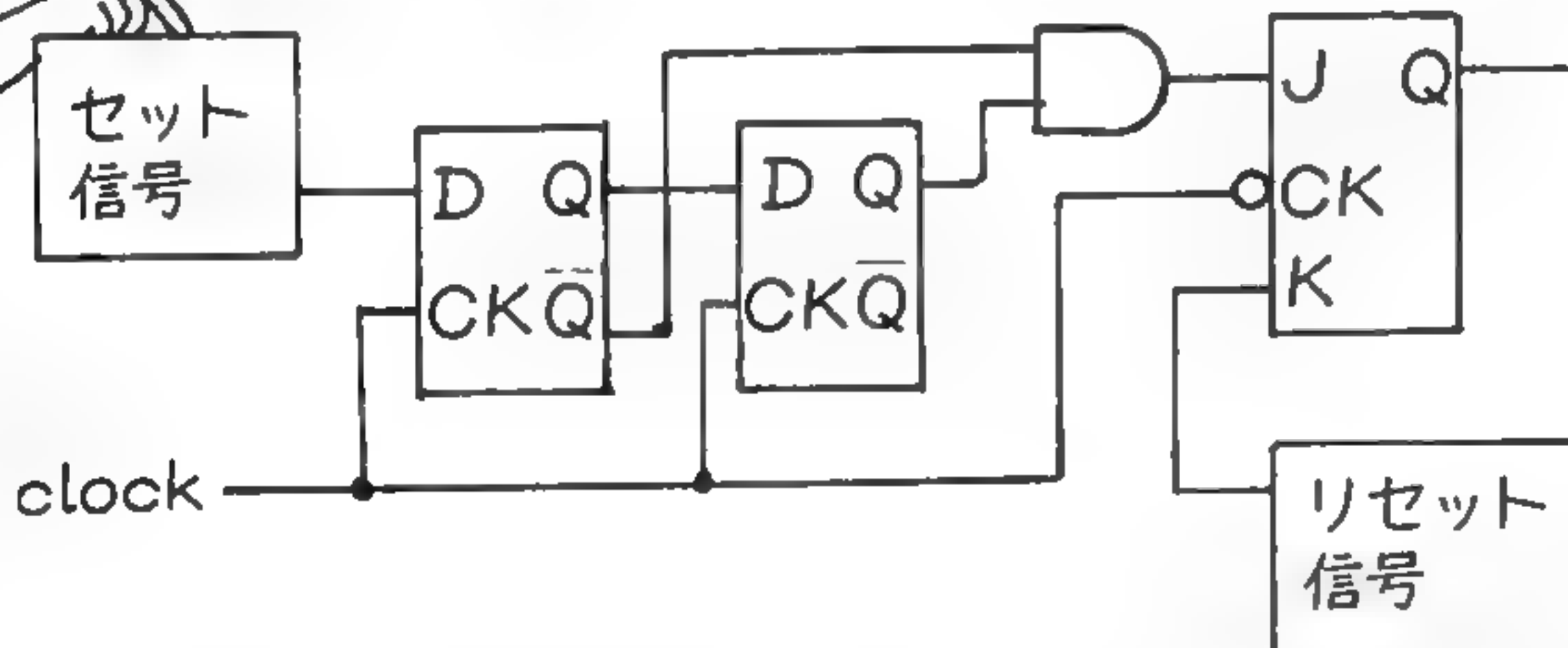
配線距離が長かったりすると、遅延や反射などでパルスの両端は、きわめて不安定な状態ですからこのような回路は信頼性がありません(回路がまったく働かないということはありません)。



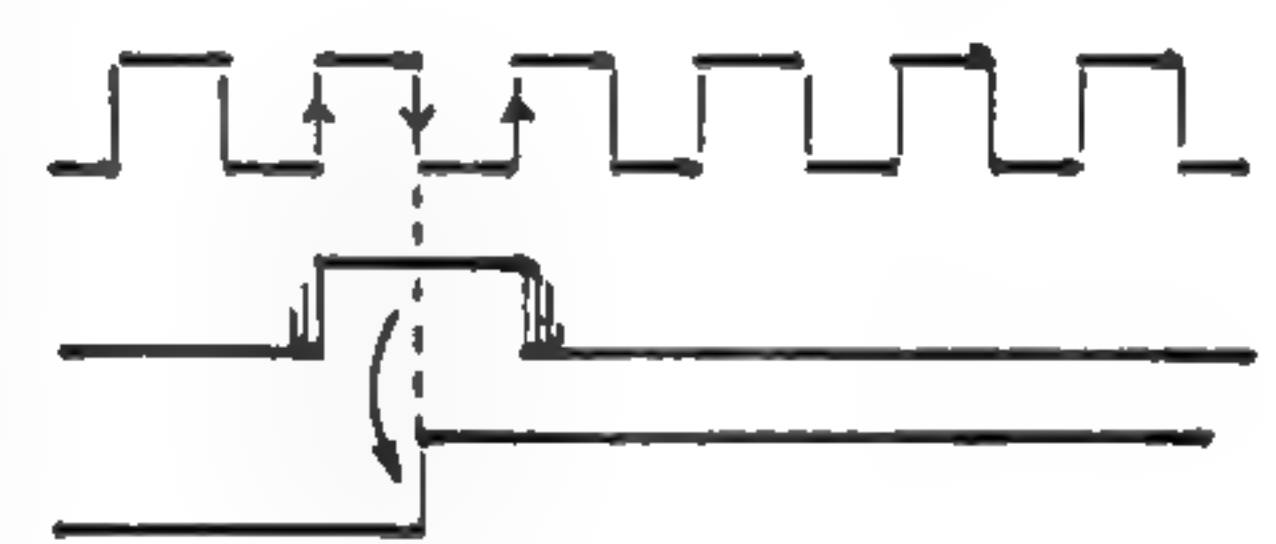
CKでつくったパルスを入れたJK FFに入れると図のようにきわどいタイミングです。



フリップフロップは1ビットのメモリと考えられます。メモリは安定域で読み書きすることが原則です。



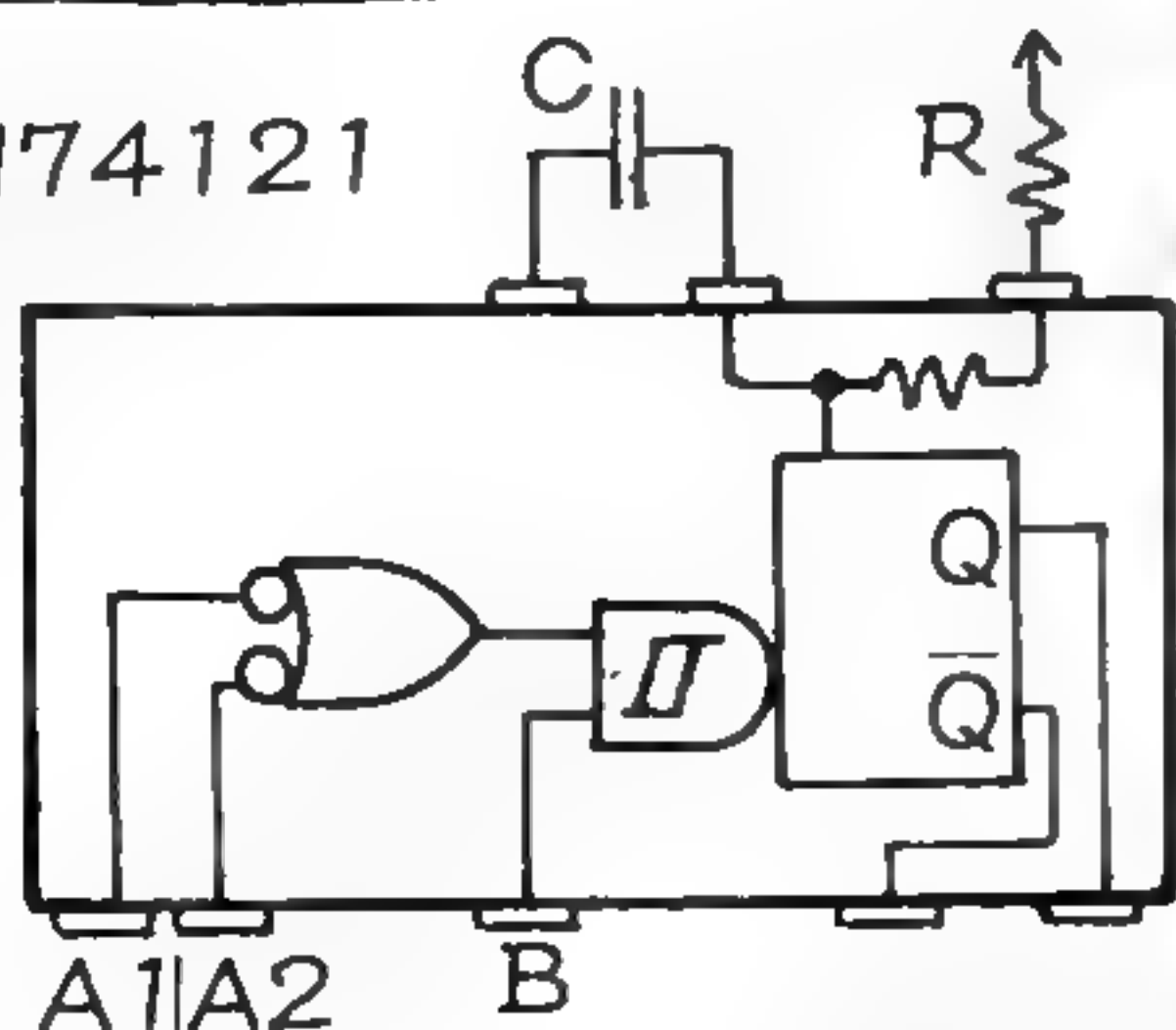
真上のコマとこのコマのタイミングの違いがわかりますか。パルスのまん中にクロックの↓がきています。



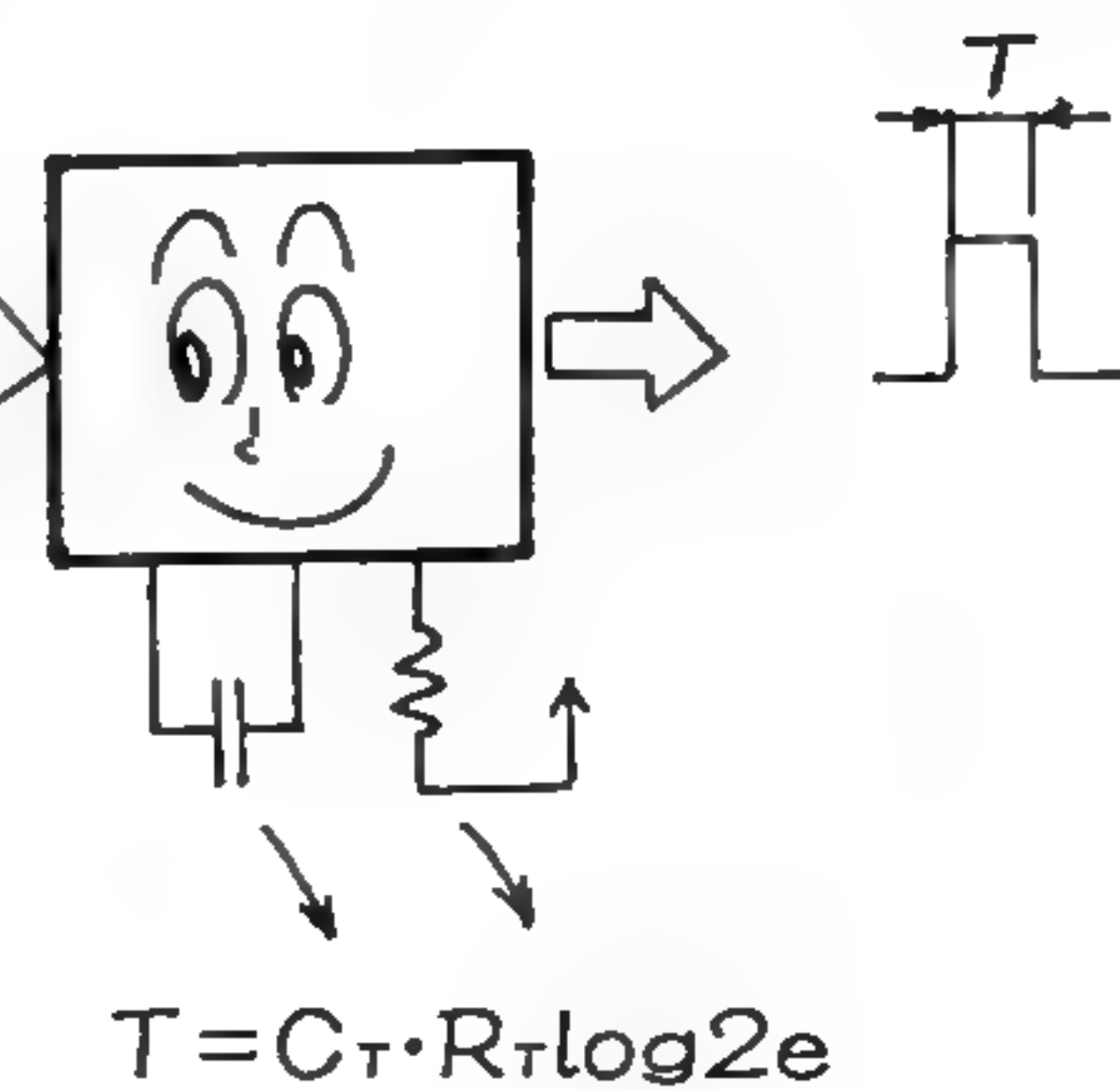
チップの中の回路はこうなっています。入力Bが↑になるとCとRで決まる長さで1パルス出力されます。

SN74121

インプット			アウト
A1	A2	B	Q
L	L	↑	パルス

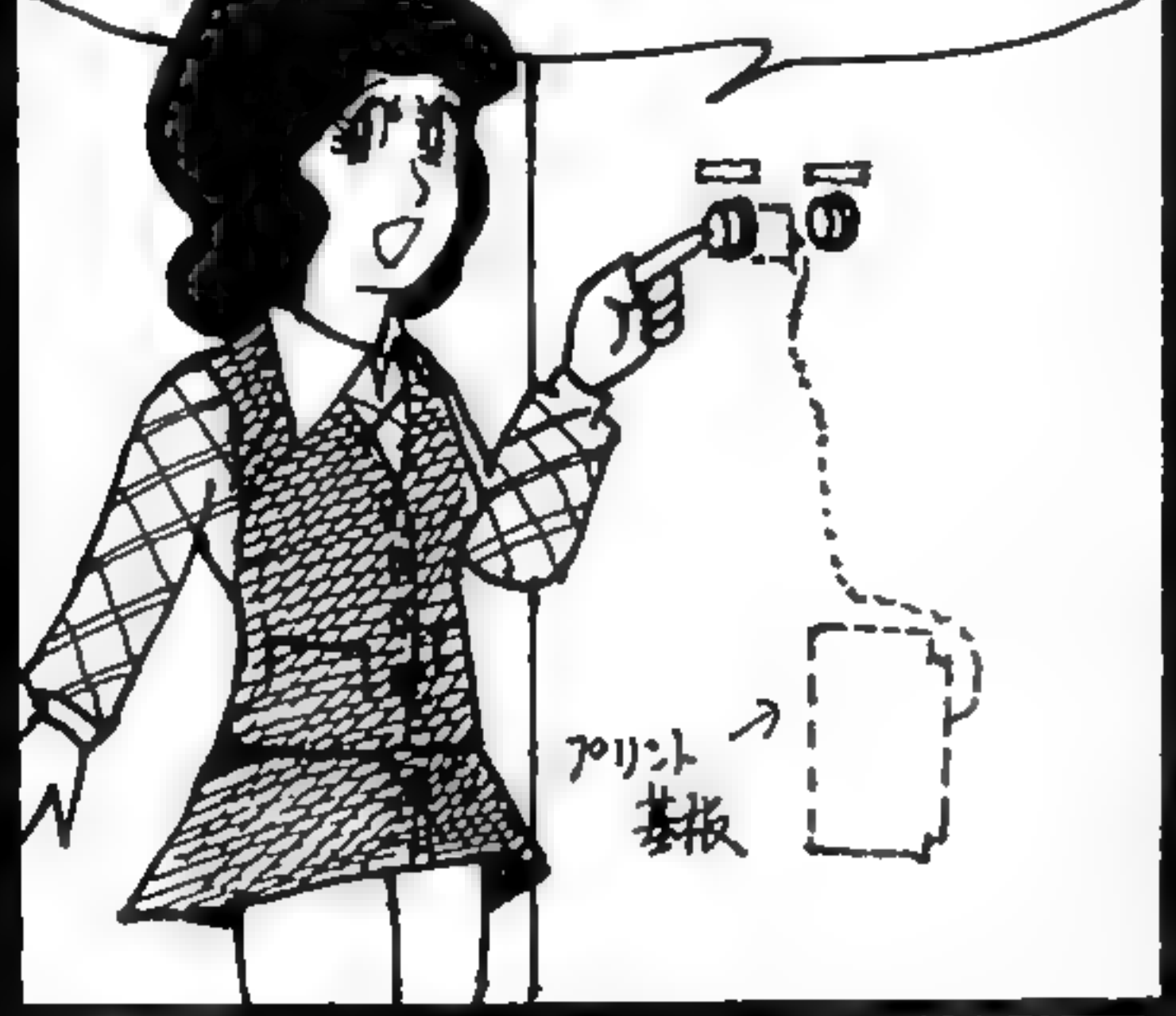


## ④ワンショットパルサー

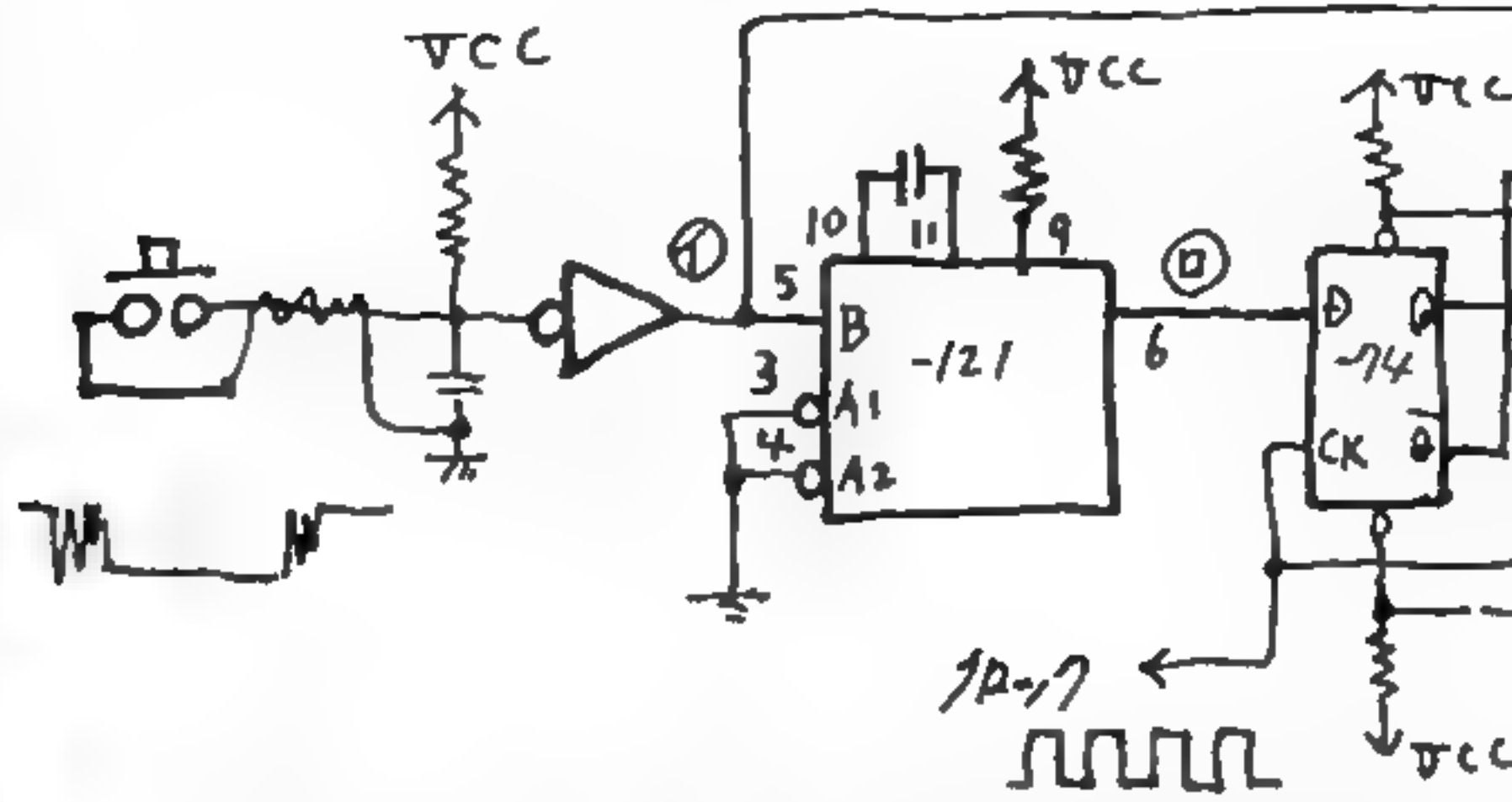




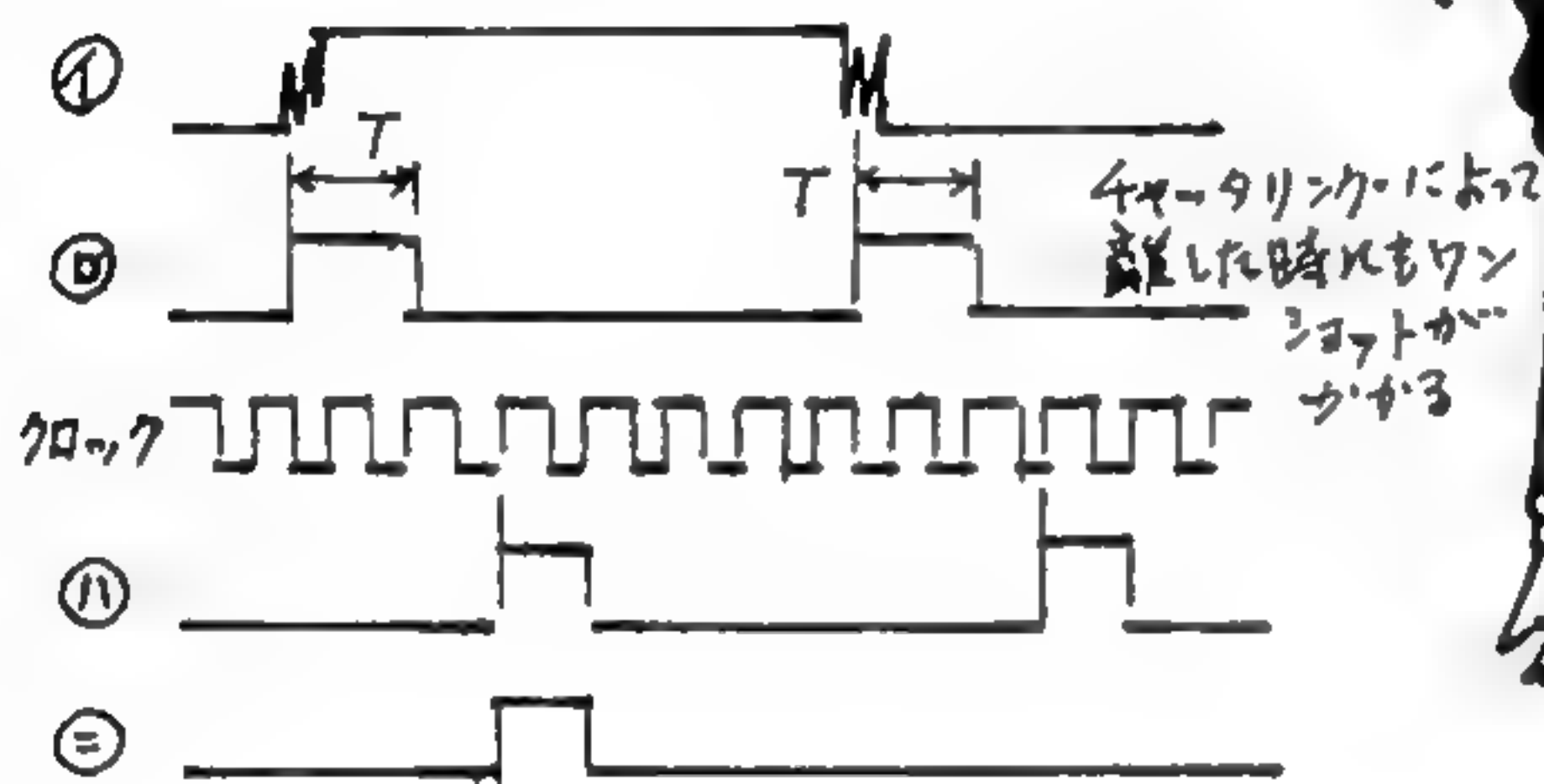
押釦はパネルの表面に取り付けられていますからプリント基板まではちょっと距離がありますから、ノイズが問題になってきます。



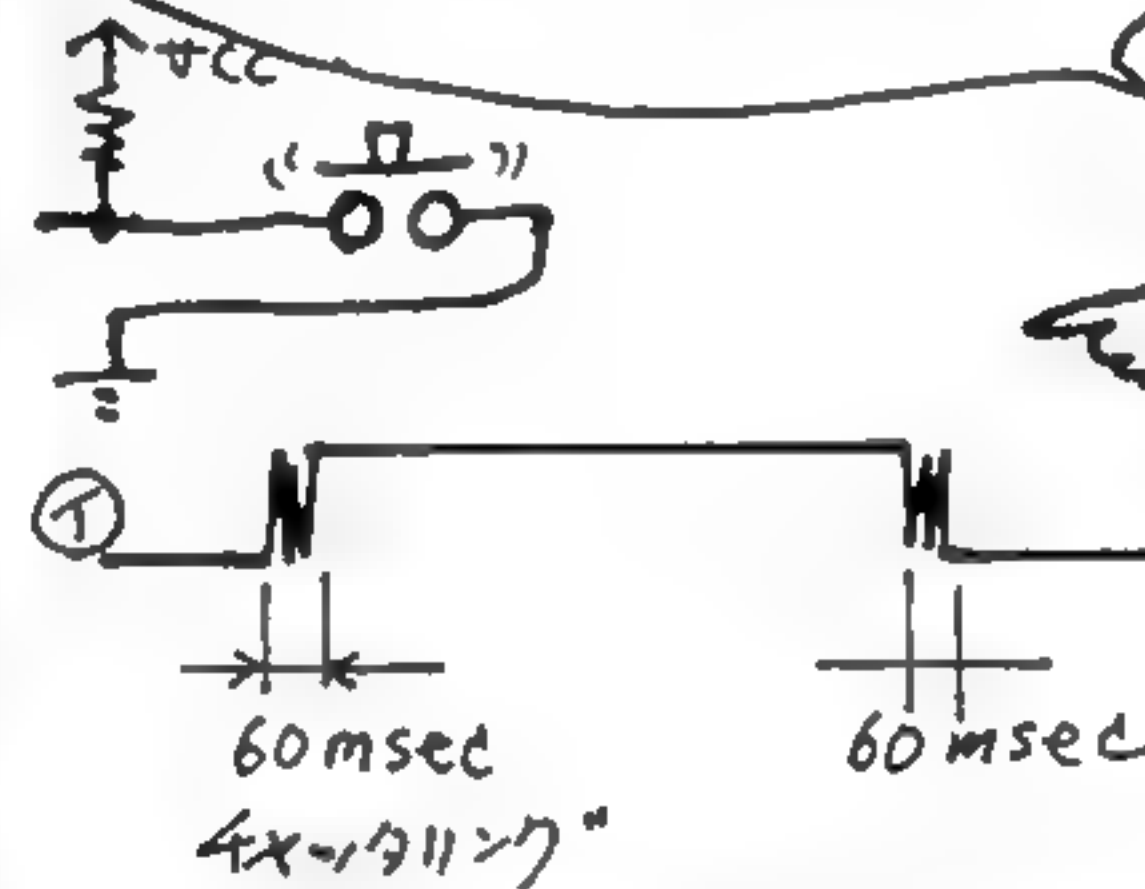
これは-121を使ったスタートパルス発生回路の一例です。押釦のチャタリングによる誤動作防止のため-121の特徴を利用しています。



スタートパルス発生回路のタイミングを見てみましょう。



スイッチを押した時やリレーの接点が入った時、またそれらの逆の時、短い時間ですが信号はON、OFFをくり返します。これをチャタリングというのです。



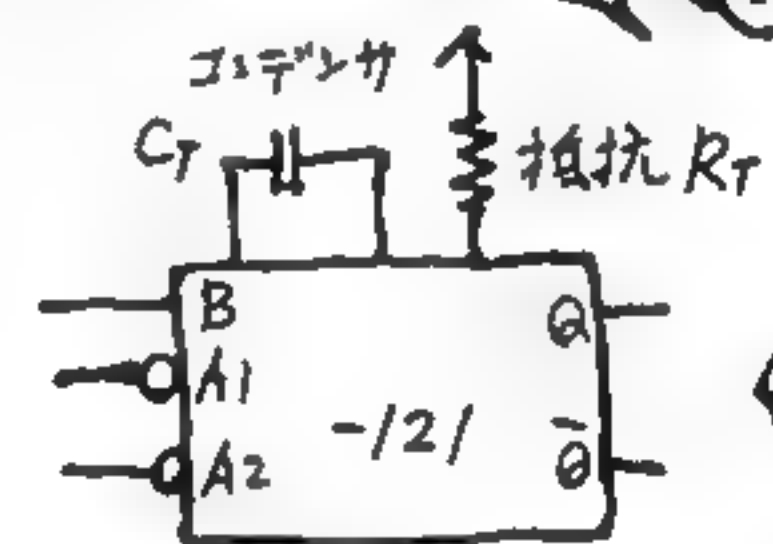
抵抗が $2K\Omega$ から $40K\Omega$ 、コンデンサが $10pF$ から $10\mu F$ までなら、 $T = C_T \cdot R_T \log 2 \approx 0.7 C_T \cdot R_T$ という関係が成立します。

例1.  $C_T = 10\mu F$   
 $R_T = 10K\Omega$  とすれば  
 $= 0.7 \times 10 \times 10 = 70 \text{ msec}$

1ms(ミリ秒)は0.001s(秒)です。

単位は便利のように $C_T$ を $\mu F$ 、 $R$ を $K\Omega$ にすれば $T$ はmsecと、なります。

SN74121でつくるワンショットのパルスによってチャタリングの時間から逃がしています。②では1回押すと1回だけのスタートパルスを得ます。このパルスの幅は、外付の抵抗とコンデンサで決定します。



時にはくつろいだひとときも必要ですよ。

フリップフロップやワンショットパルサーはマイコン出現まではIC回路になくてはならないものですが...

タイミングをとるための抵抗とコンデンサは-121の近くにとりつけます。

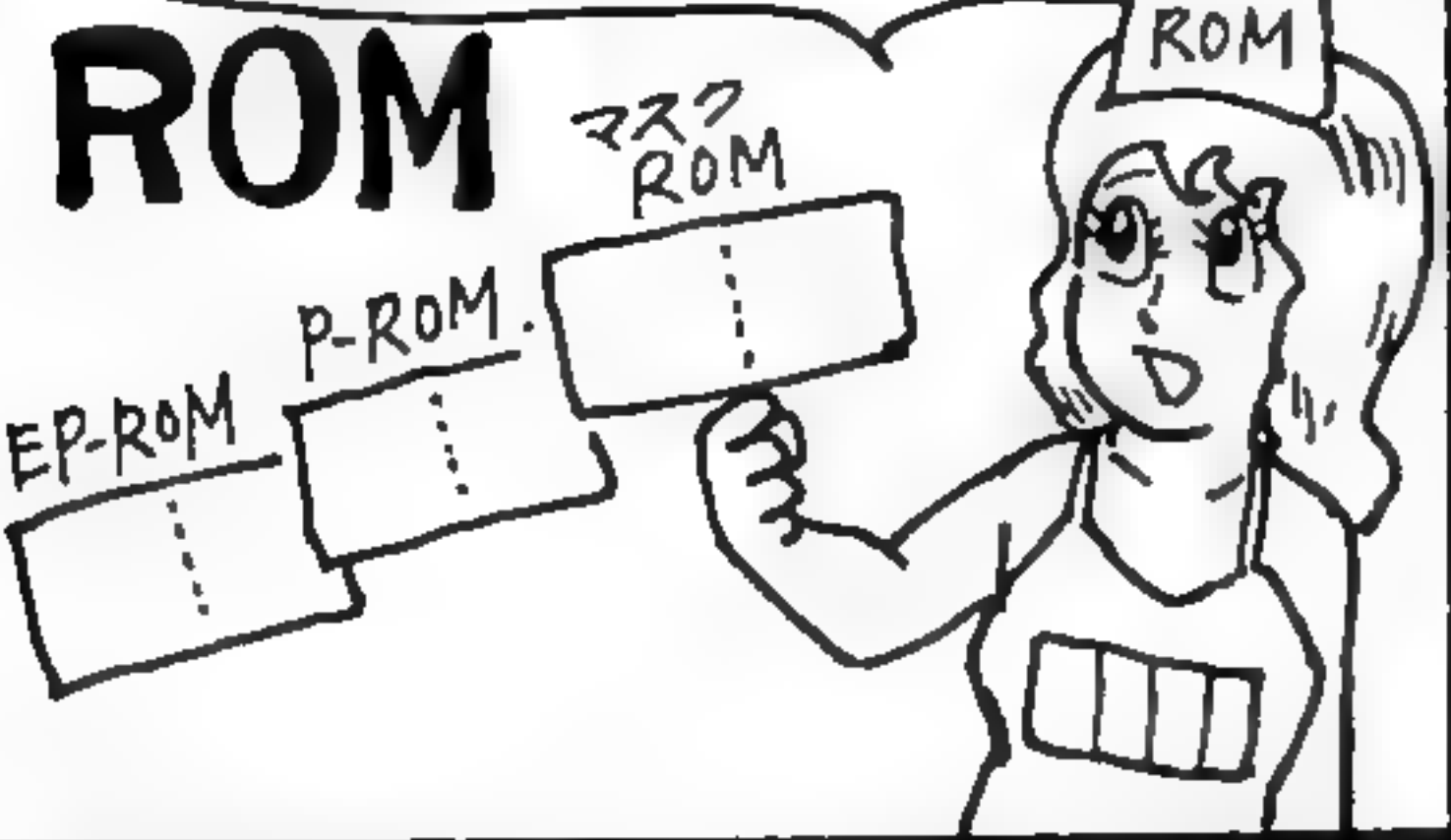
10 $\mu F$  10K $\Omega$

約70msのパルスが出来ます。

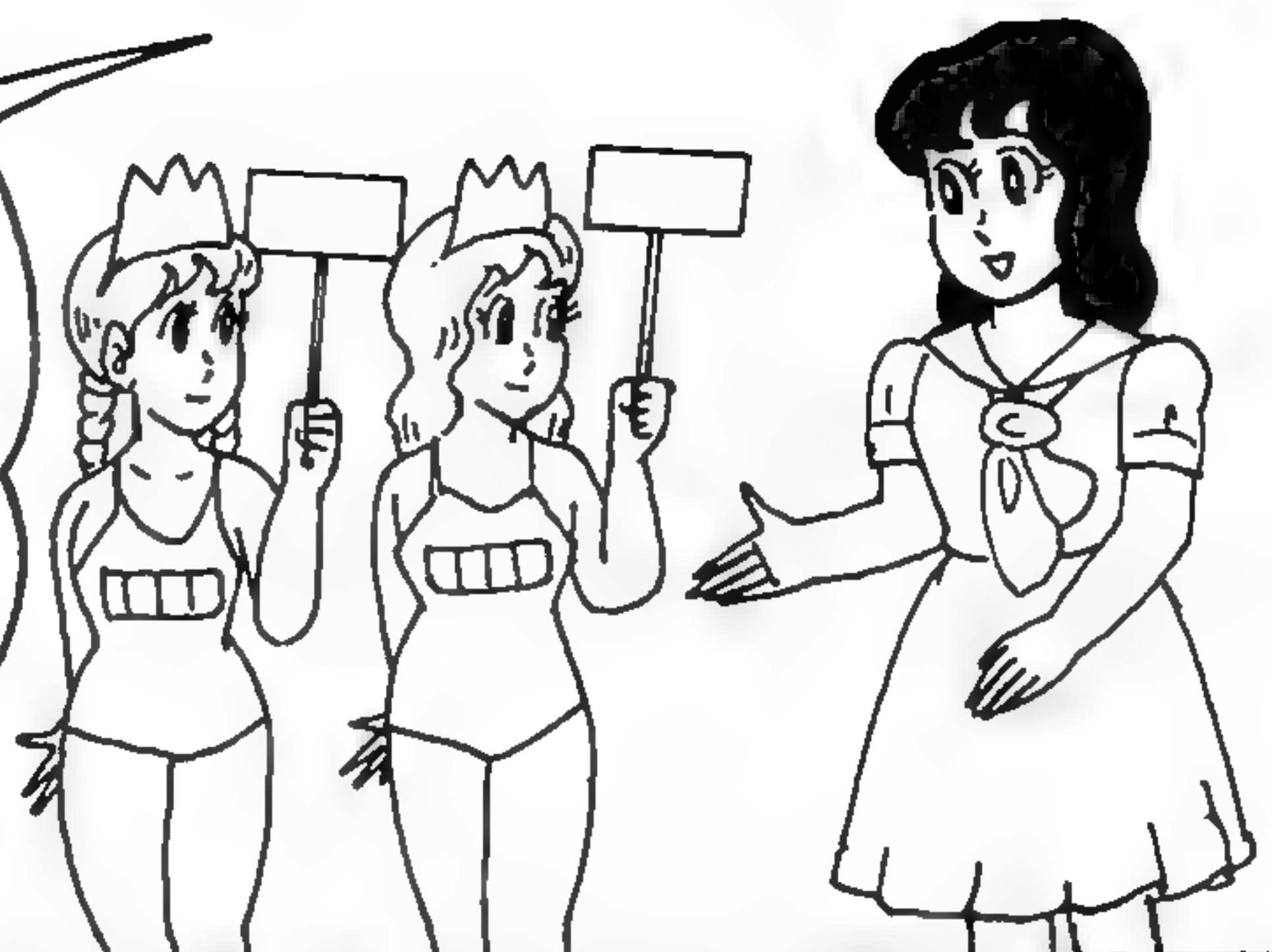


## ⑤ROMとRAM

まず、ROMちゃんです。ROMちゃんはデータの読み出し専用メモリです。ということは、あらかじめ何らかの方法でデータが書き込まれていなければなりませんね。その書き込み方から見て3種のROMがあります。



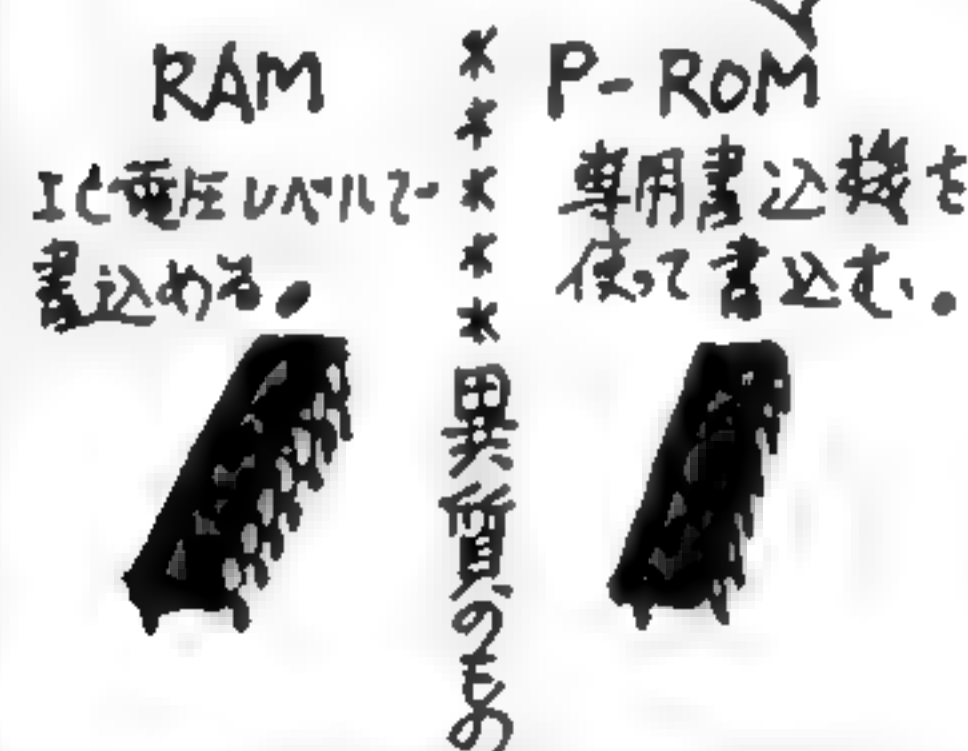
ROMちゃんとラムちゃんはこれまで何度となく登場しましたがハード的なことをもう少しお話ししましょう。



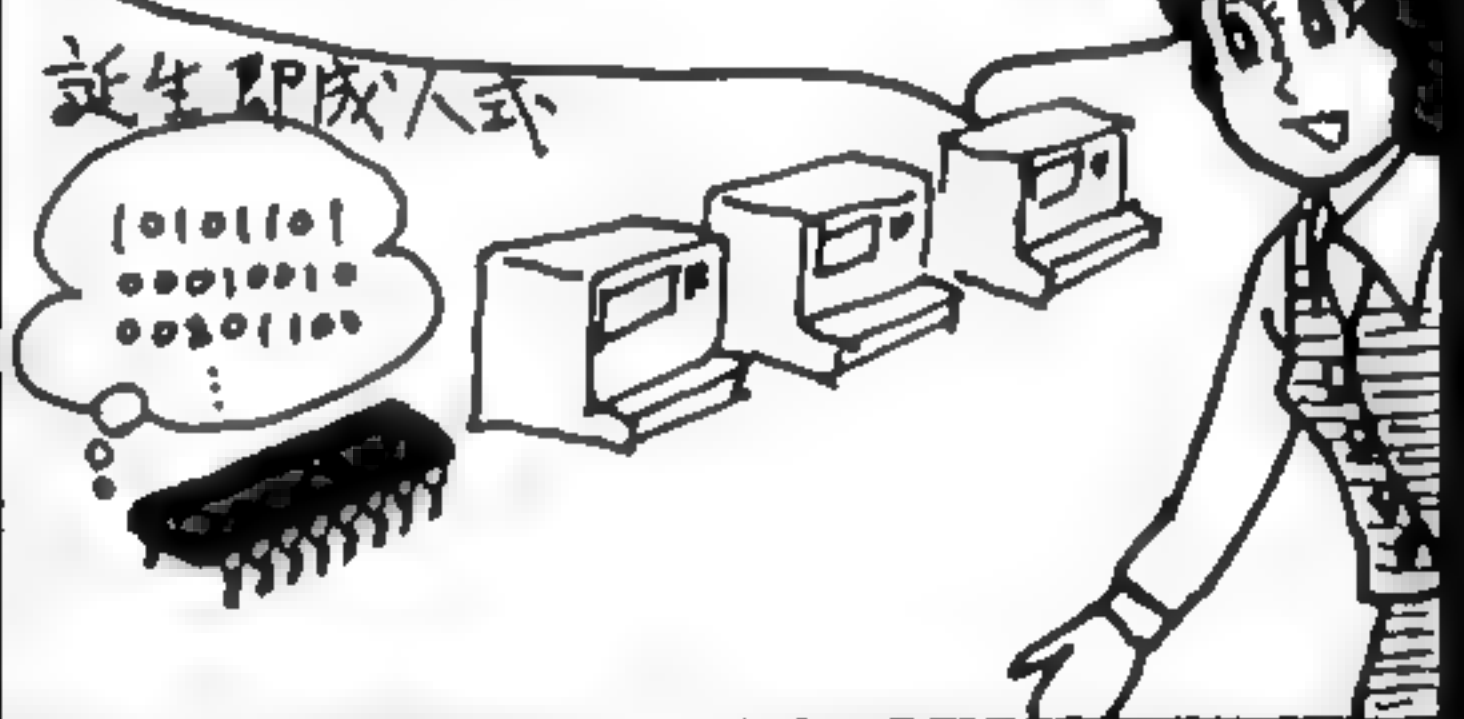
そのプログラムも回路の一部を破壊することによって作るので一回書き込んだら変更はできないのです。失敗したらそのROM1個オシャカということになります。



P-ROMは、プログラムできるROMの略です。これはユーザーが自分のプログラムを書き込み機を使ってデータを入れられるということです。RAMへの書き込みとは次元が違うものです。



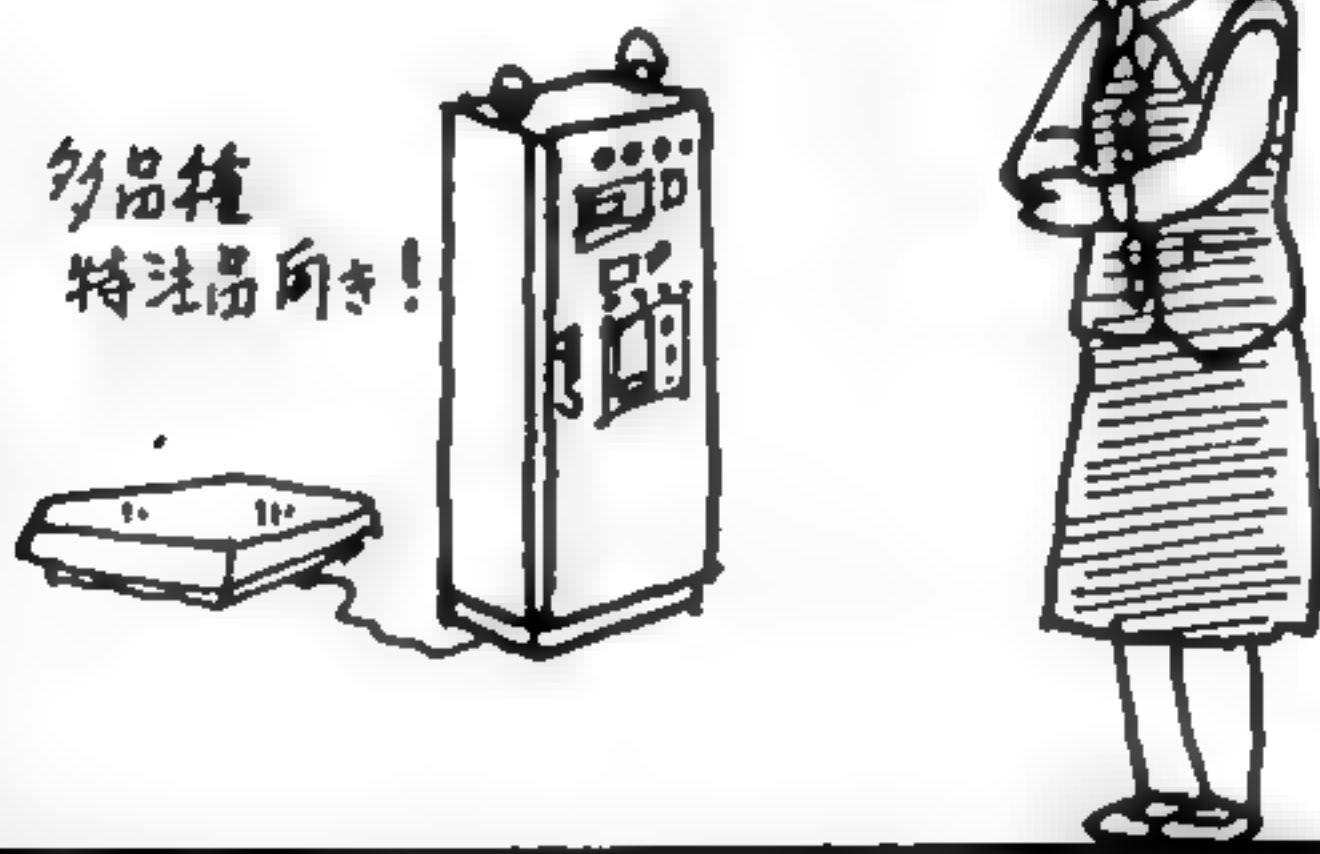
マスクROM、これは量産製品に組み込む場合に適しています。ROM製造過程でプログラムされているので改めて書き込む手間が省けます。そのかわり他の製品に使用することはできません。



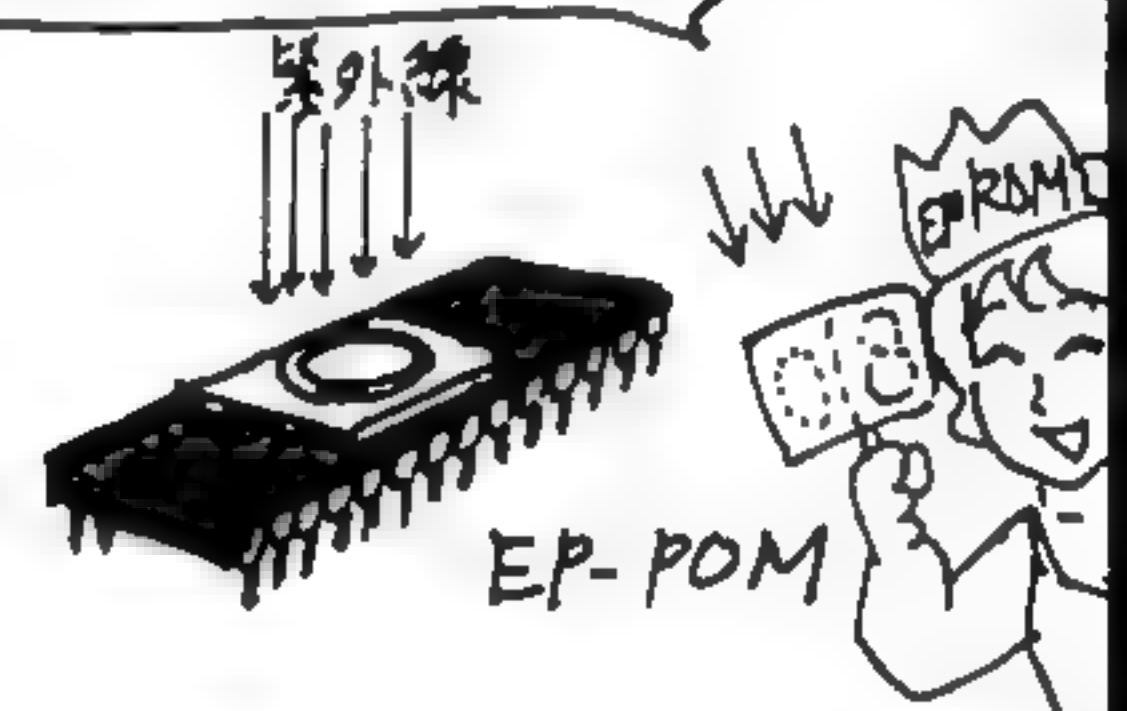
- ROMを書き込み方法で分けると……
- ☆マスクROM…製造過程でプログラム量産品向き
  - ☆P-ROM……ユーザーがプログラム。一度書き込むと修正できない
  - ☆EP-ROM……ユーザーがプログラム。消去用エネルギーを加えれば再度プログラム可能

この3種類になります。

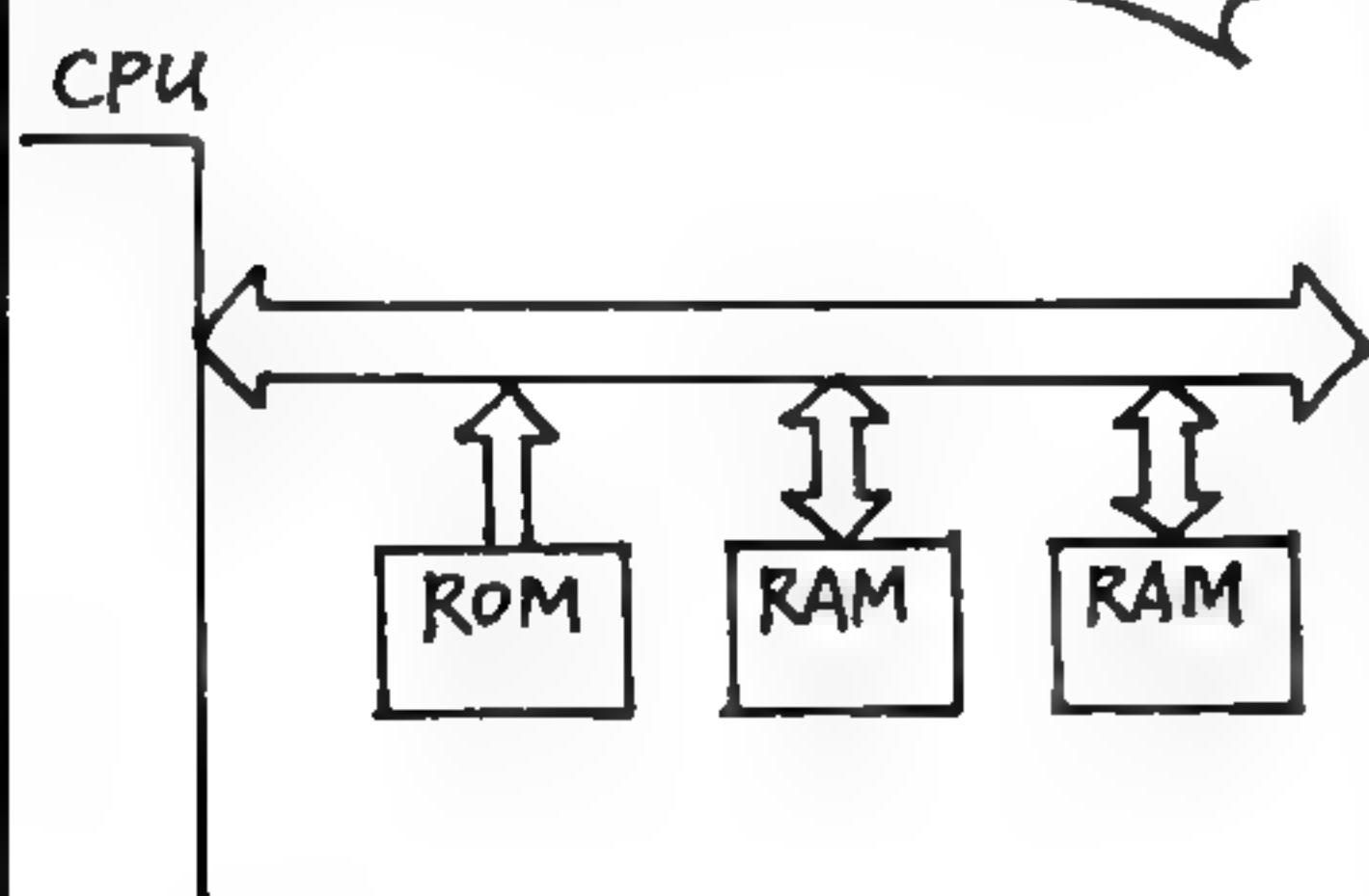
紫外線を照射することによって消すものが一般的でしたが、このごろは電氣的に消せるものもあるようです。このEP-ROMの出現によってROMのムダ使いがなくなりました。



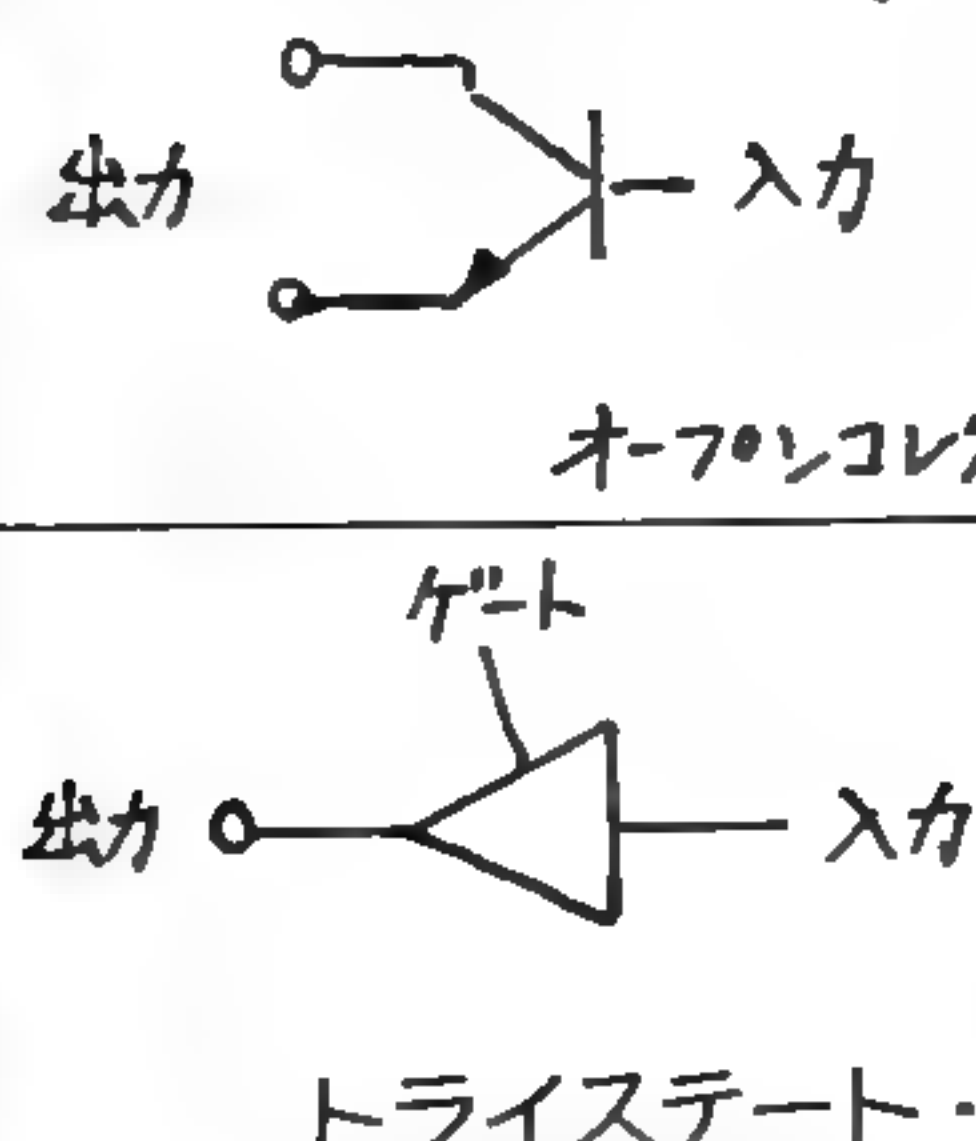
そこで失敗したり仕様変更になってオシャカになるはずのROMをもう一度よみがえらせるようにしたのが、3つ目のEP-ROMというものです。Eはイレイザブルつまり書き込んだデータを消すことができ、元の状態に戻せるものです。



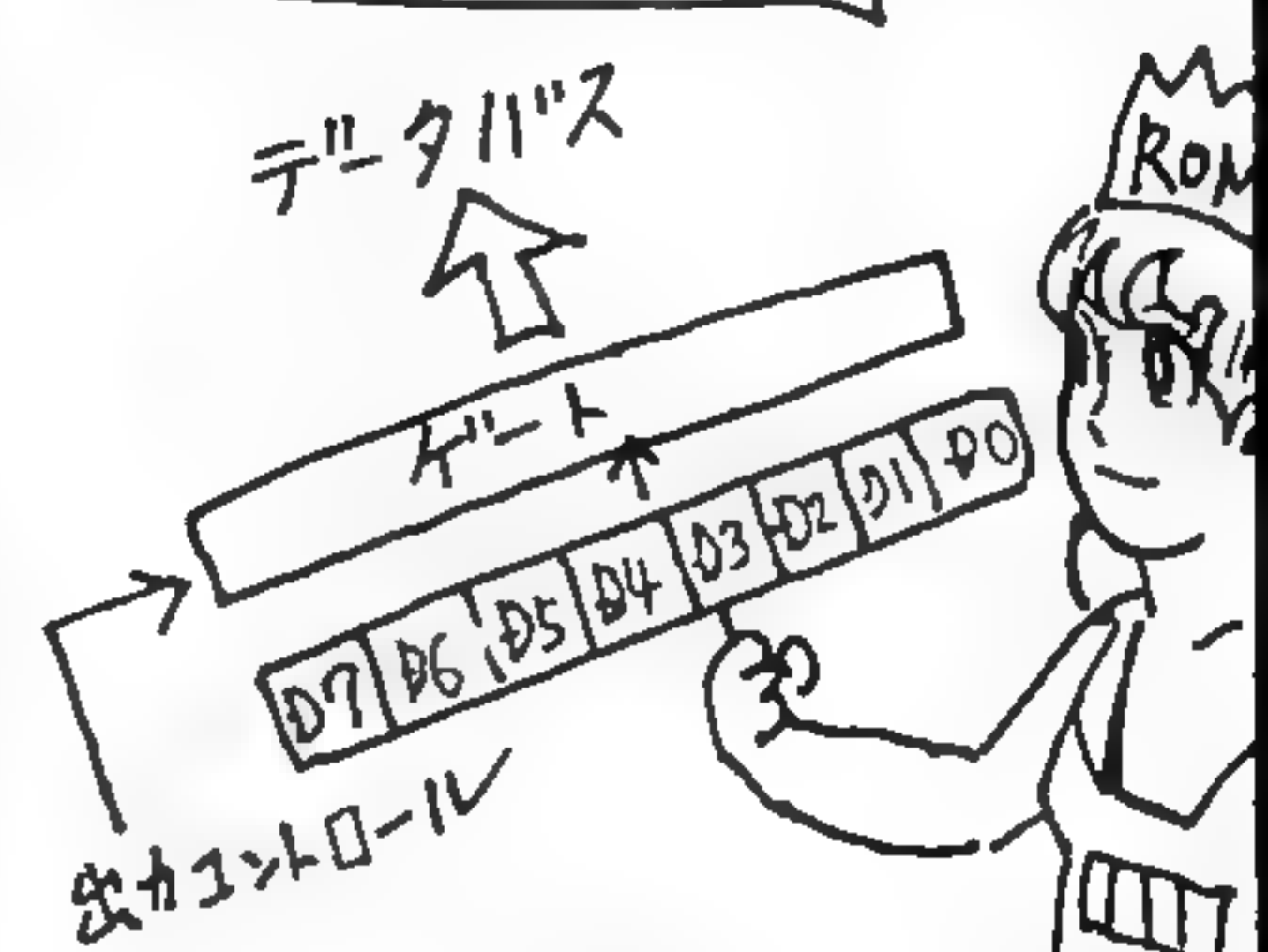
オープンコレクタのものは応答速度の関係で今ではあまり使えないようです。トリステート・ゲートについてはもうすぐ出てきます。次はRAMちゃんです。



そのため、ROMのデータ側はオープンコレクタ、またはトリステート・ゲートになっています。



ROMちゃんのデータは、データバスに対して出力として働くのでセレクトされない時はバスに影響を与えないものでなくてはなりません。





リフレッシュというわずらわしさはありますが、大容量のメモリが安価にできるという利点がありますので、パーソナルコンピュータに向いています。

RAM 64 Kバイト



ダイナミックRAMは記憶素子がコンデンサなので一定時間ごとにデータを読み出して再び書き込むというリフレッシュ回路が、必要なのです。



RAMちゃんはメモリ保持の方法で次の2つに分けられます。ひとつはダイナミックRAMで、他のひとつはスタティックRAMです。

# RAM



8080Aや8085AはスタティックRAMしか使えませんが、Z-80にはリフレッシュ用のカウンタがついています。

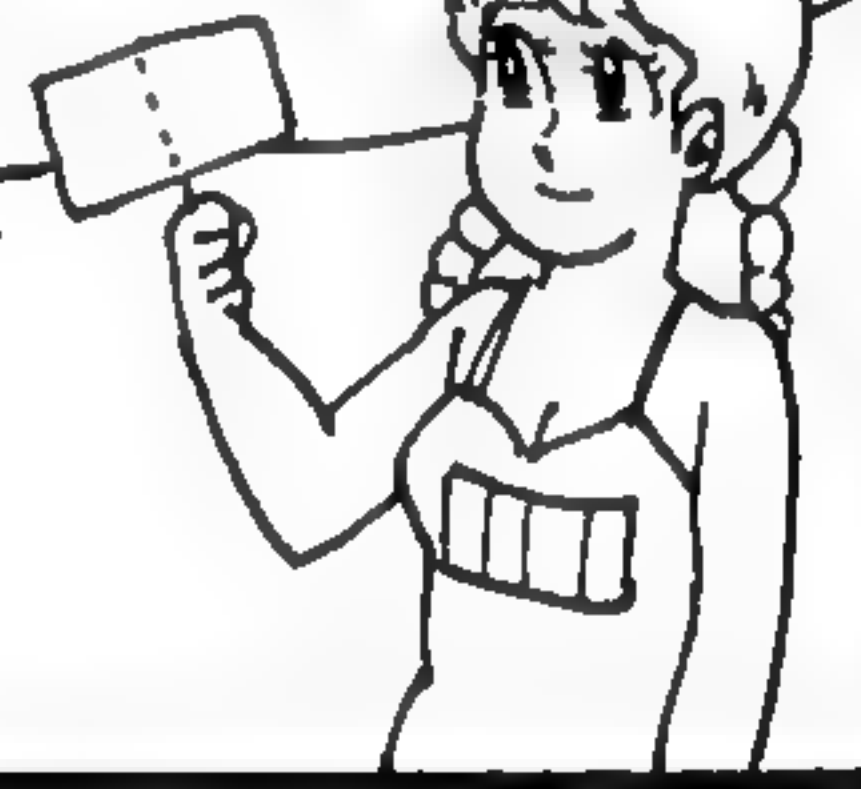


それから、ダイナミックRAMのアクセス時間は10nsecぐらいで、スタティックRAMとは1桁違います。

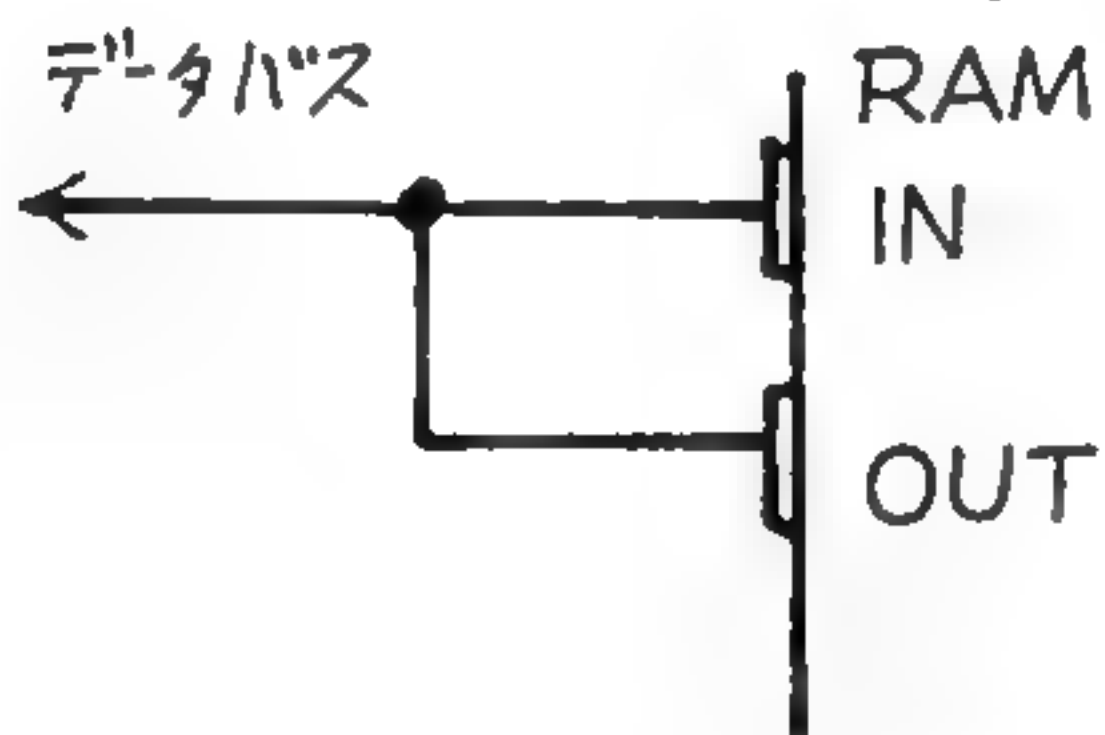
制御用とか、ちょっとしたデータ処理用秤量機なんかは、スタティックRAMを使っています。



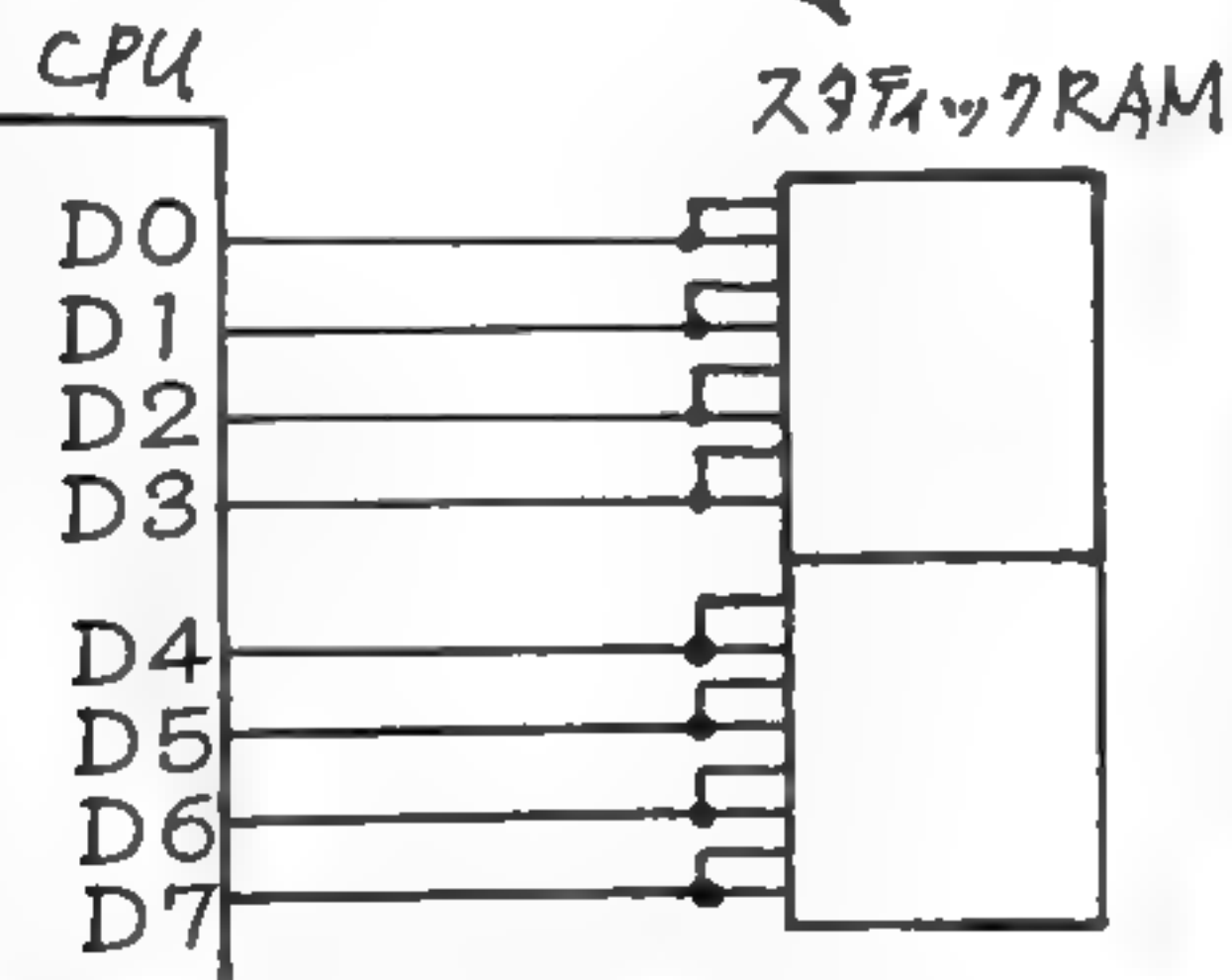
一方スタティックRAMはトランジスタで双安定を保つため、リフレッシュという回路のわずらわしさはありません。ただメモリのバイト当りの単価が高いのです。4 Kバイト以下のマイコンに多く使われています。



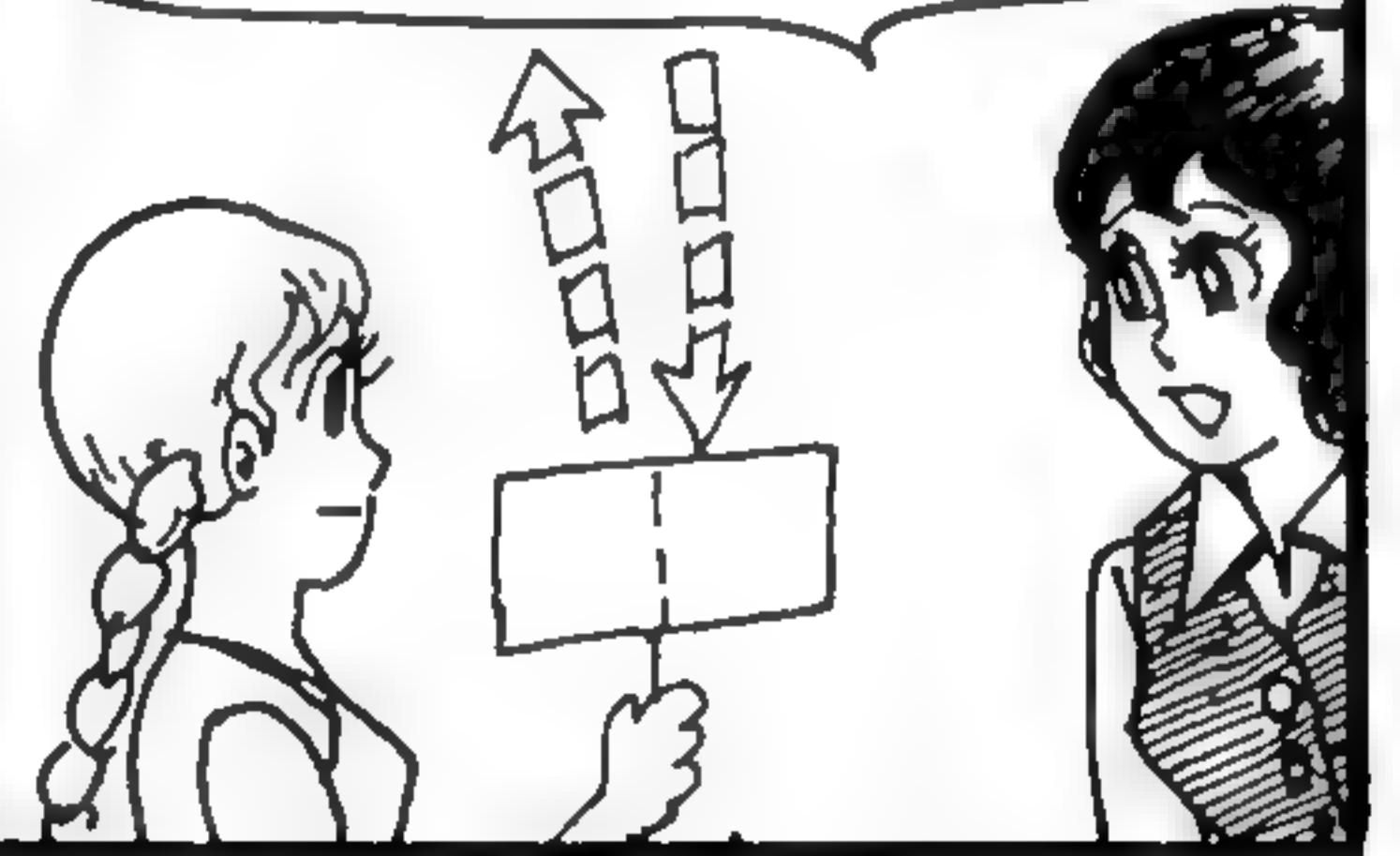
回路図をよく見ると、RAMのデータピンはIN、OUT 2種のものがあります。1種のはパッケージの中の回路でつないでいるのです。



データバスは8ビットです。1ビットは1本の導線で伝達でき特殊なものではありません。



データでROMちゃんとは違うのはRAMちゃんのデータは双方向性だということです。データバスの信号の方向がCPUからのコントロールによって変わります。



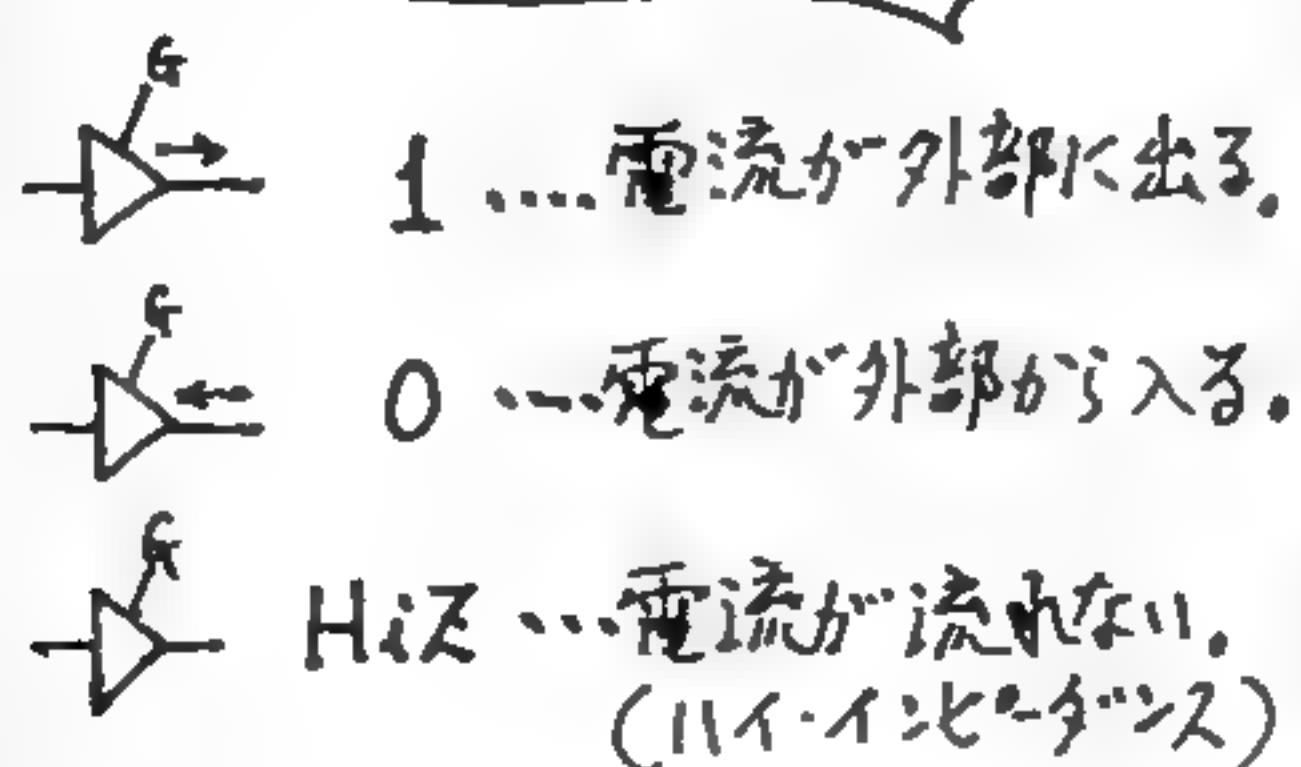
それからRAMちゃんの場合とはくに読み出し、書き込みにそれぞれ時間がかかるということを考えておかねばなりません。では、これでRAM、ROMメモリの説明はおしまいです。

書き込み ... 250 nsec Min  
Write Pulse

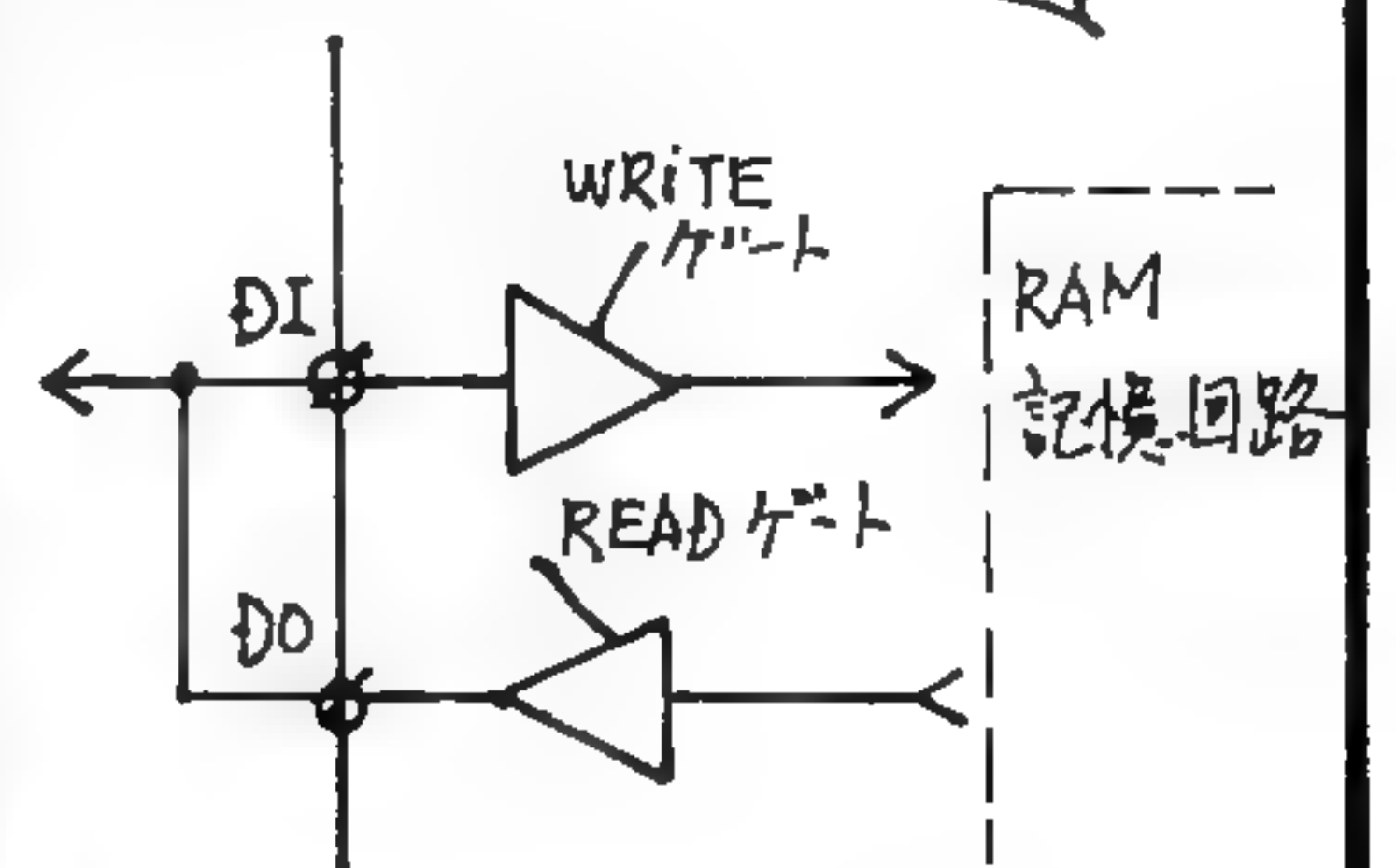
読み出し ... 450 nsec Max  
Access Time

スタティックRAM  
8101A-4の場合

同時にゲートをあけるようなことをしなければ出力側を並列につないでも回路が混乱しないようにできているのがトライステート・ゲートです。

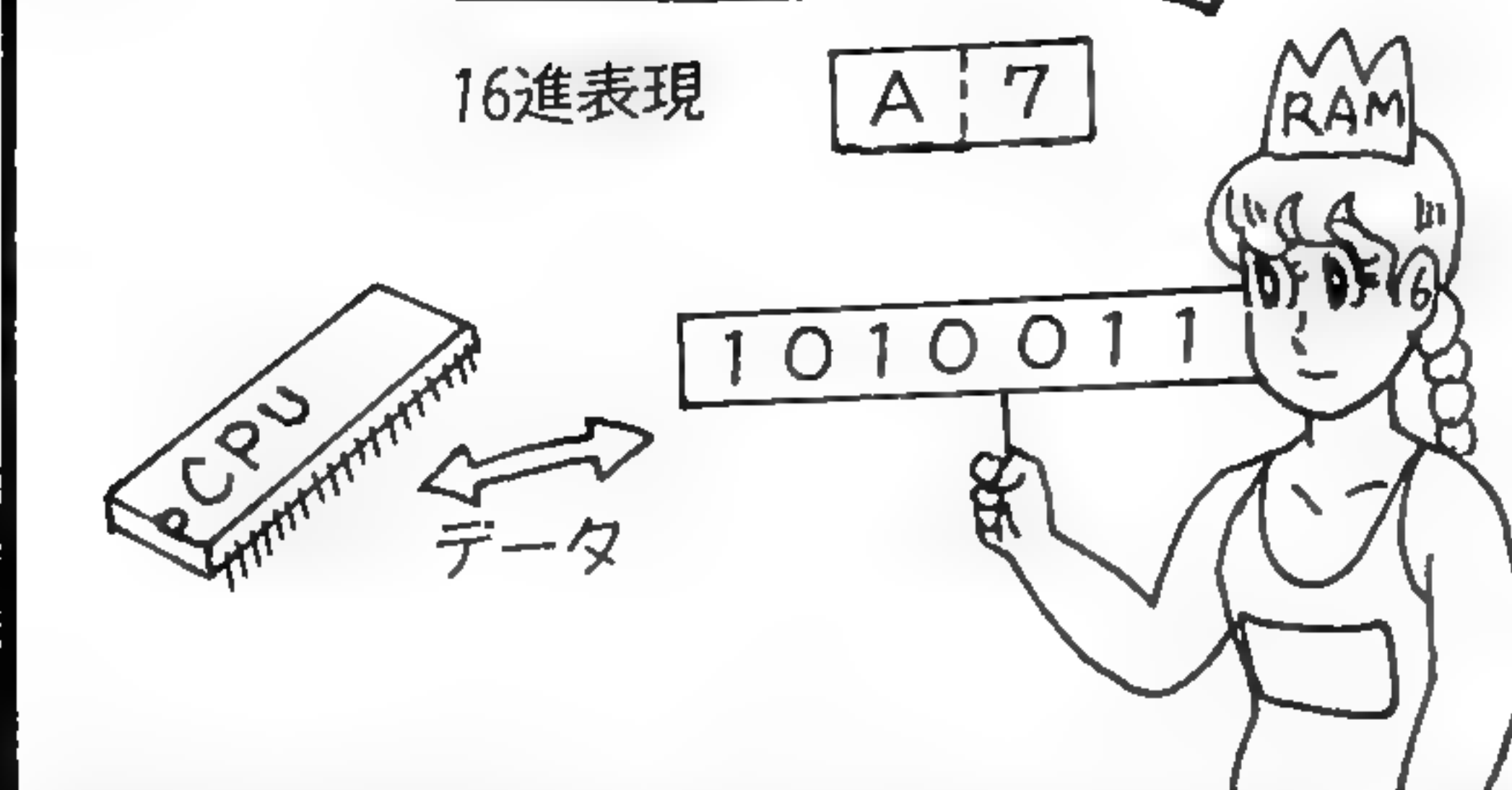


この中はどうなっているのかというと、トライステート・ゲートがIN、OUT用にひとつずつあってRead/Write信号でコントロールしているのです。

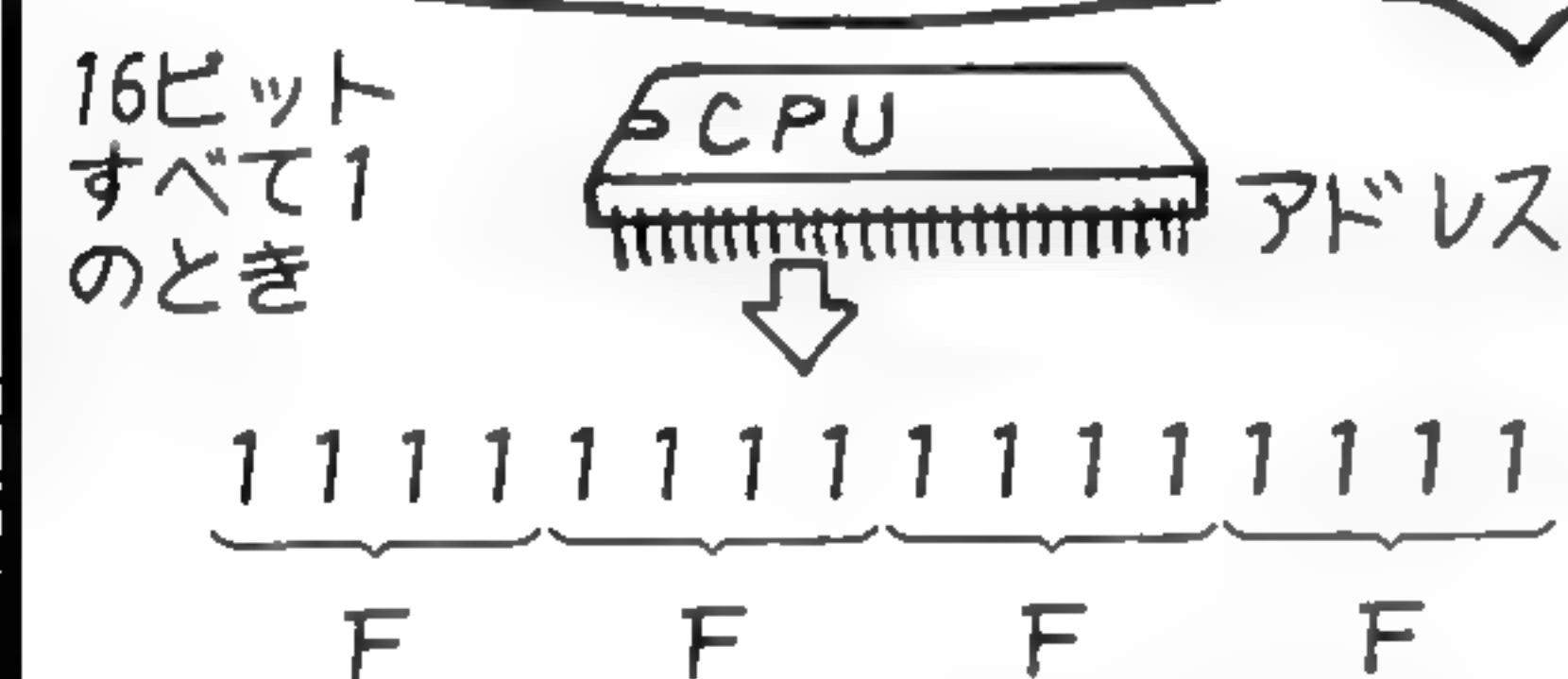




マイコンで8ビットCPUといっているのは、データとして1度に8ビット読み書きができるCPUということなのです。



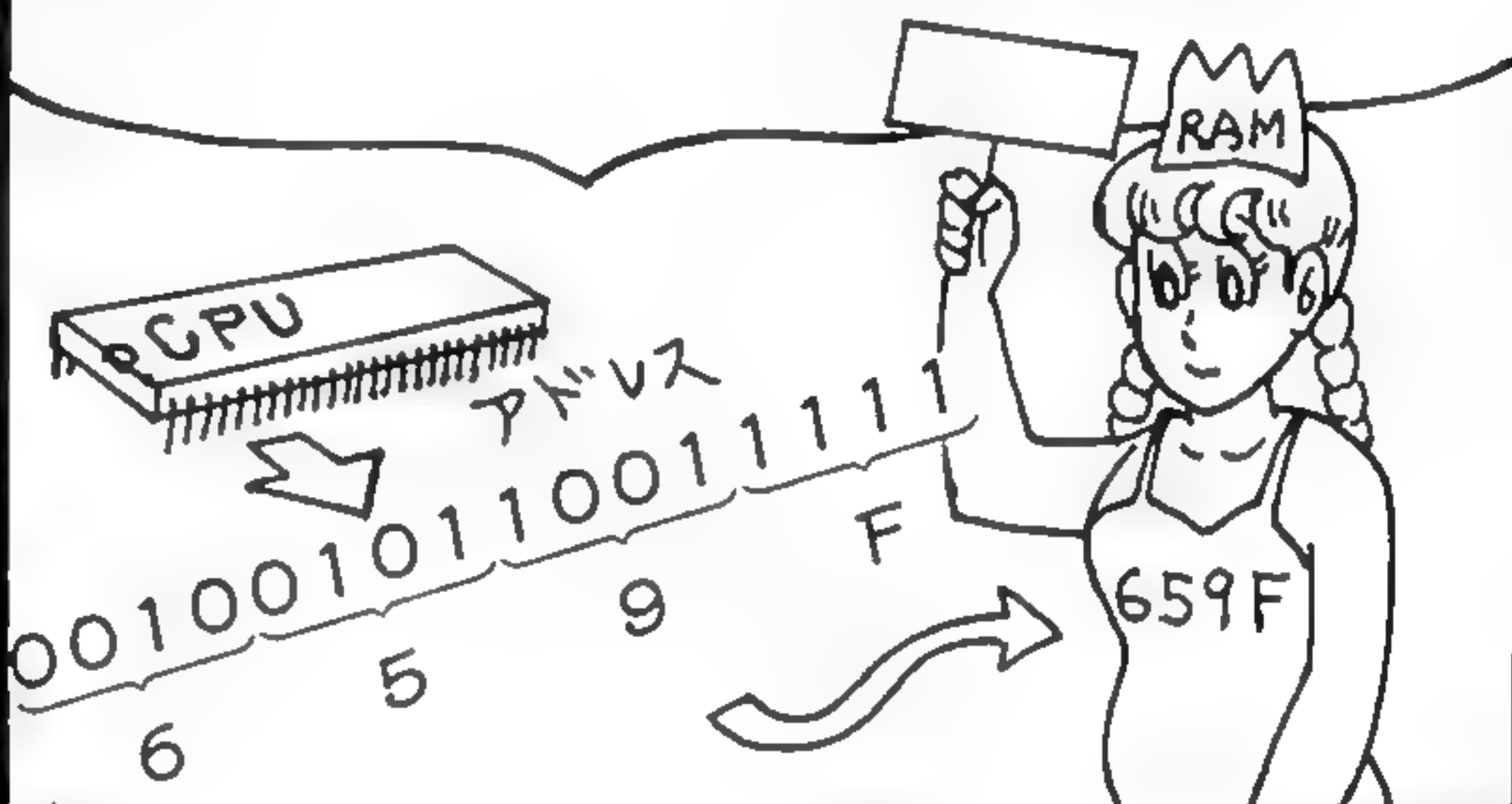
この16ビットで指定できる数は0000～FFFFFまでで10進数では65536通りです。普通パソコンのメモリは最大65K(キロ)バイトといわれるのはここからきています。



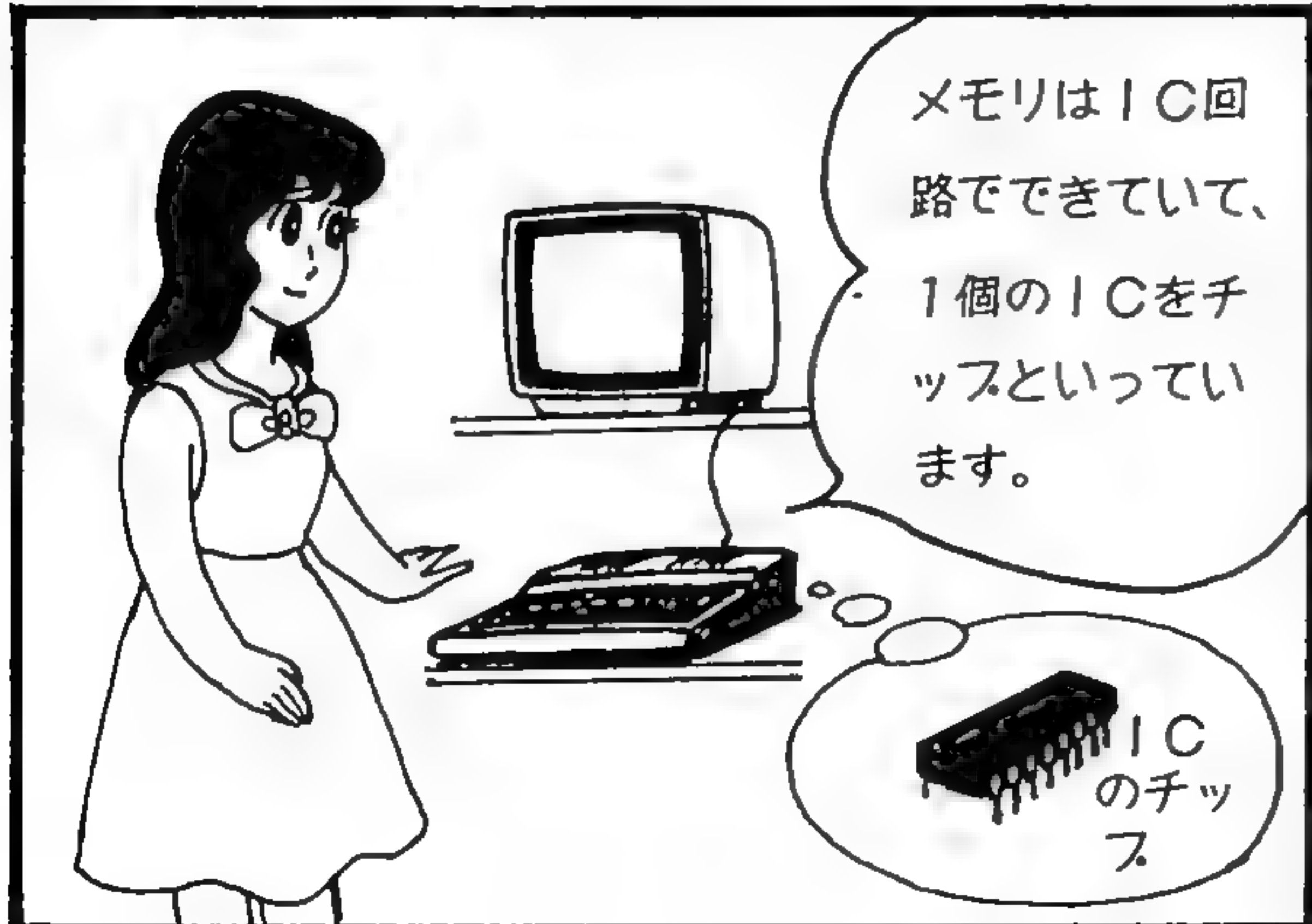
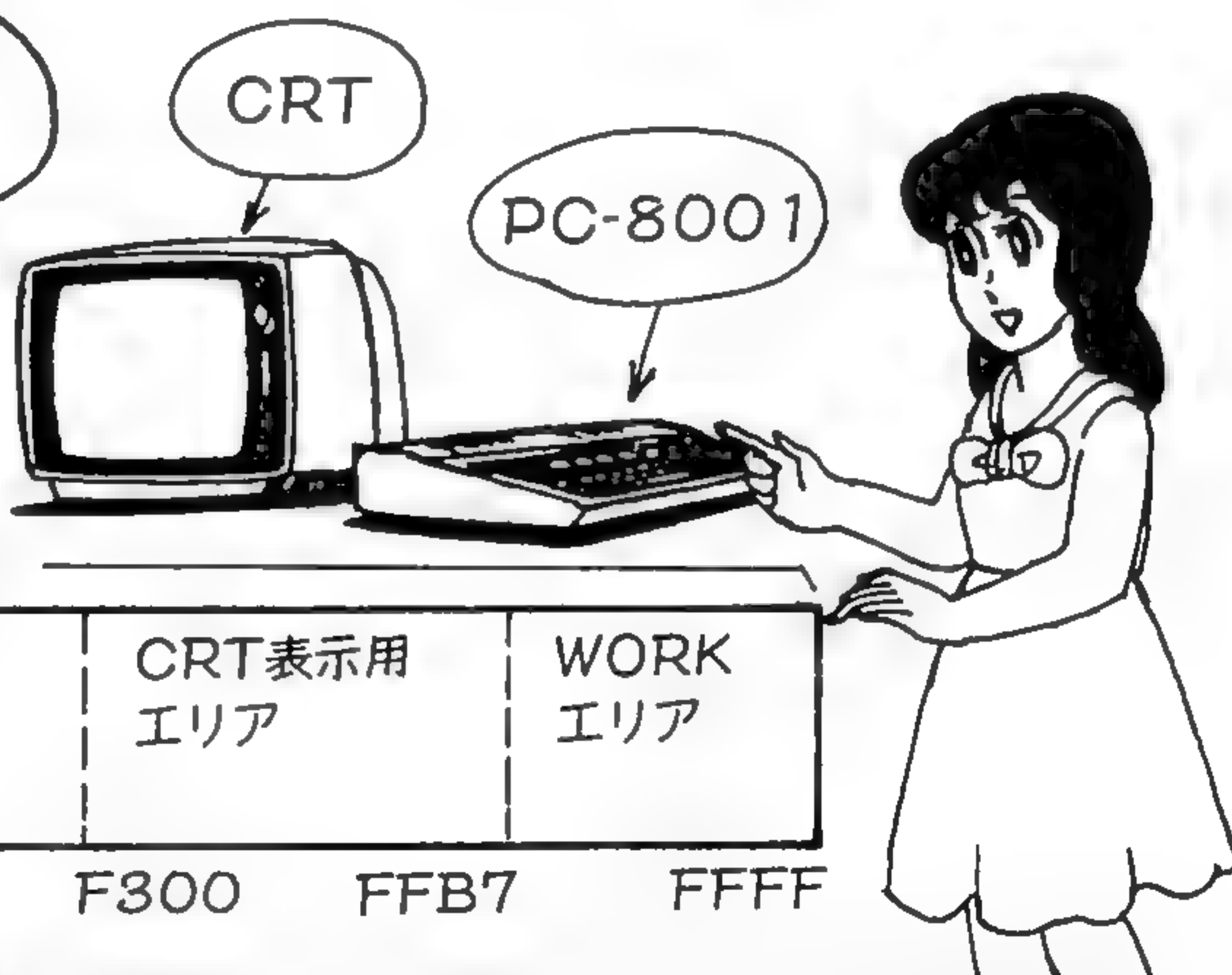
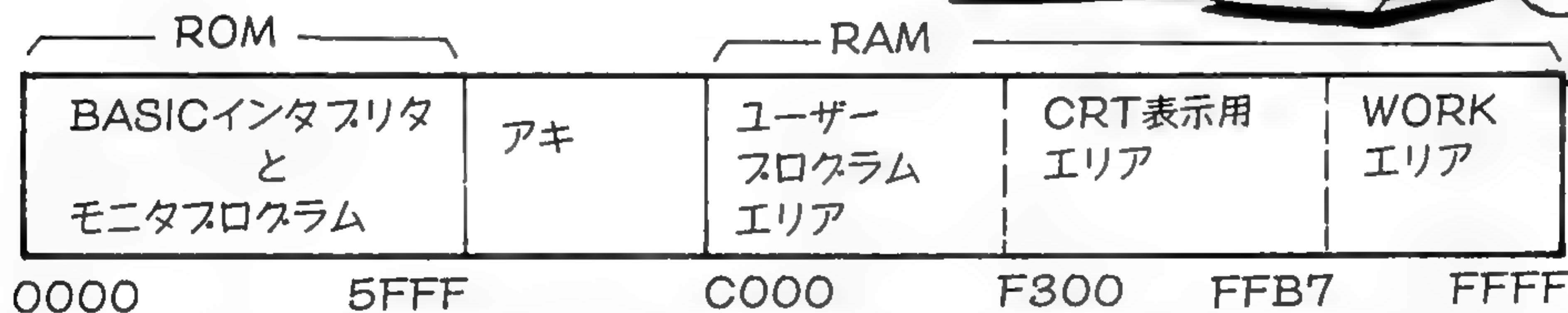
## ⑥メモリ容量とチップセレクト



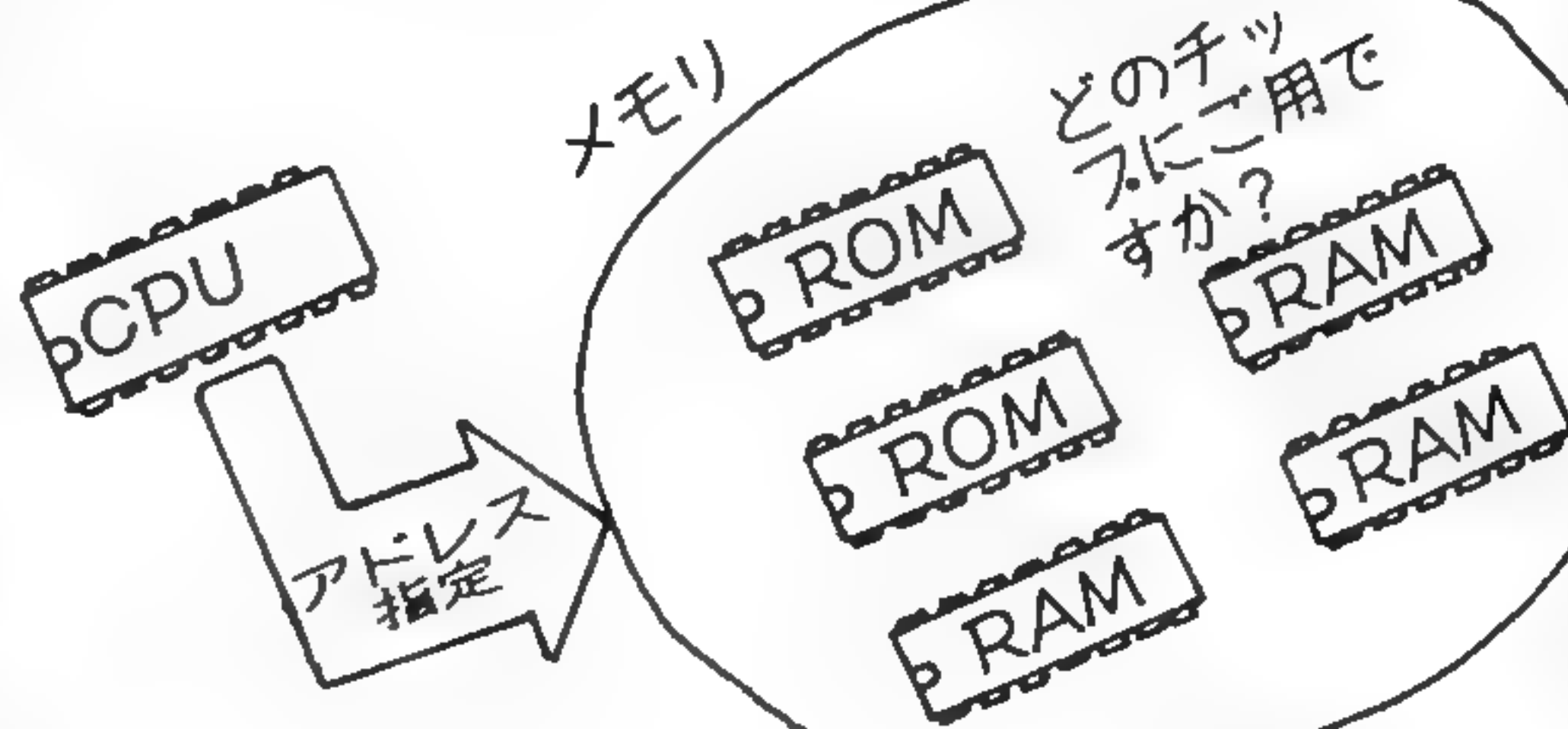
その読み書きするデータがどこにあるかはっきりさせるのがアドレスです。8ビットCPUでは16ビットのアドレス出力があります。



このアドレスは製作時点で決まっています。たとえばPC-8001のメモリアドレスマップはこのようになっています。



ところで使っているメモリはROMやRAMの種類があり同じRAMでも何個か組み合わせなければならない場合があります。





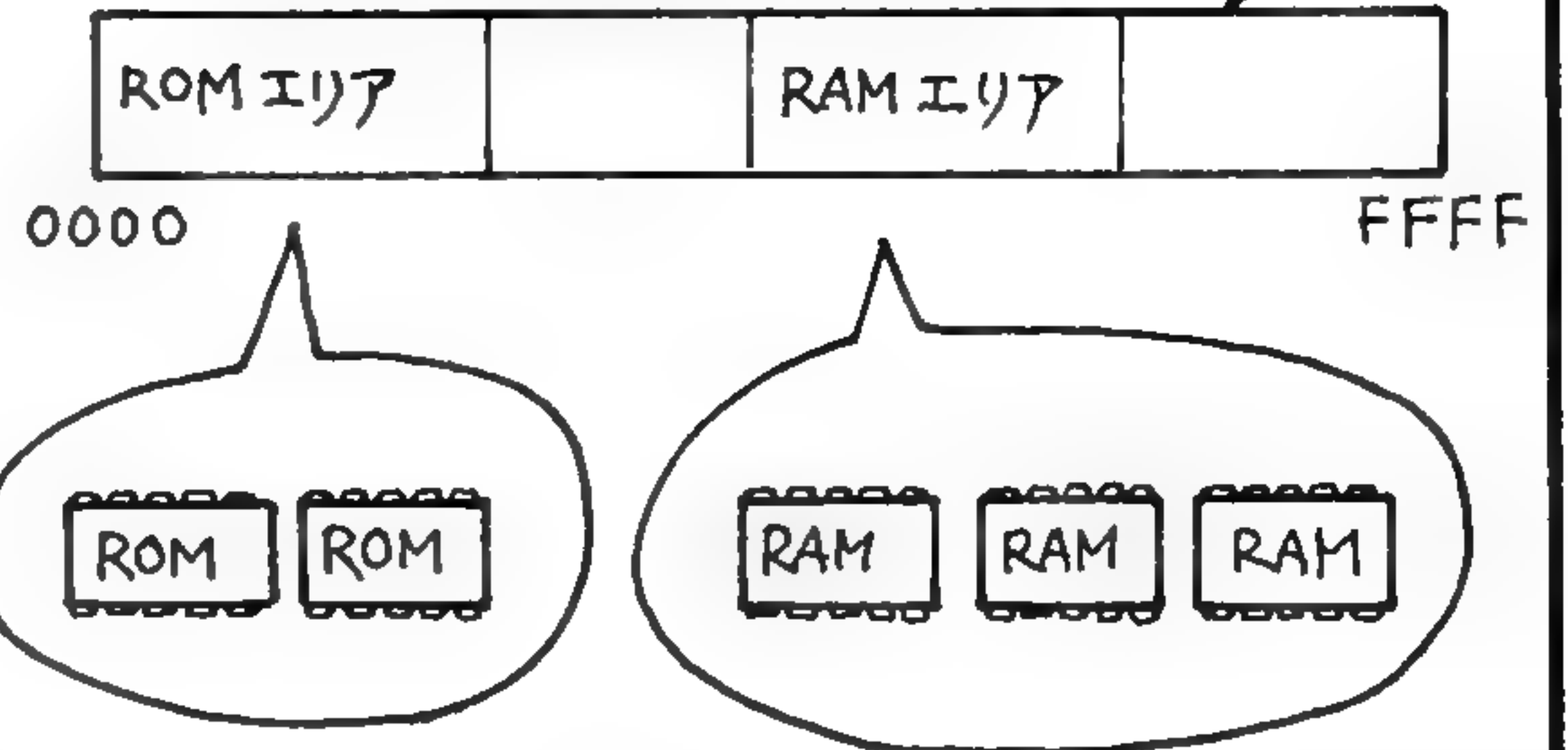
実際にはこんなバイト数のメモリはありませんが、今8バイトのメモリがあるとします。ROM、RAMはこの場合関係ありません。



私たちがプログラムする時には考える必要のないことですが、何個かのチップをどうやってつなぐのか参考までにお話しましょう。

メモリマップは何個かのチップを組み合わせで作られるはず…。

CPUからみたメモリマップ



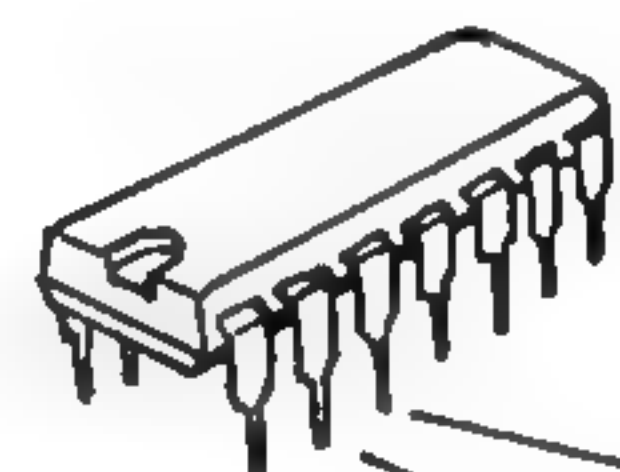
たとえば110という信号をアドレスピンに入れてやると6番のメモリを指定したことになります。



アドレスピンに入る信号と選ばれる1バイトのメモリの関係はこうなります。

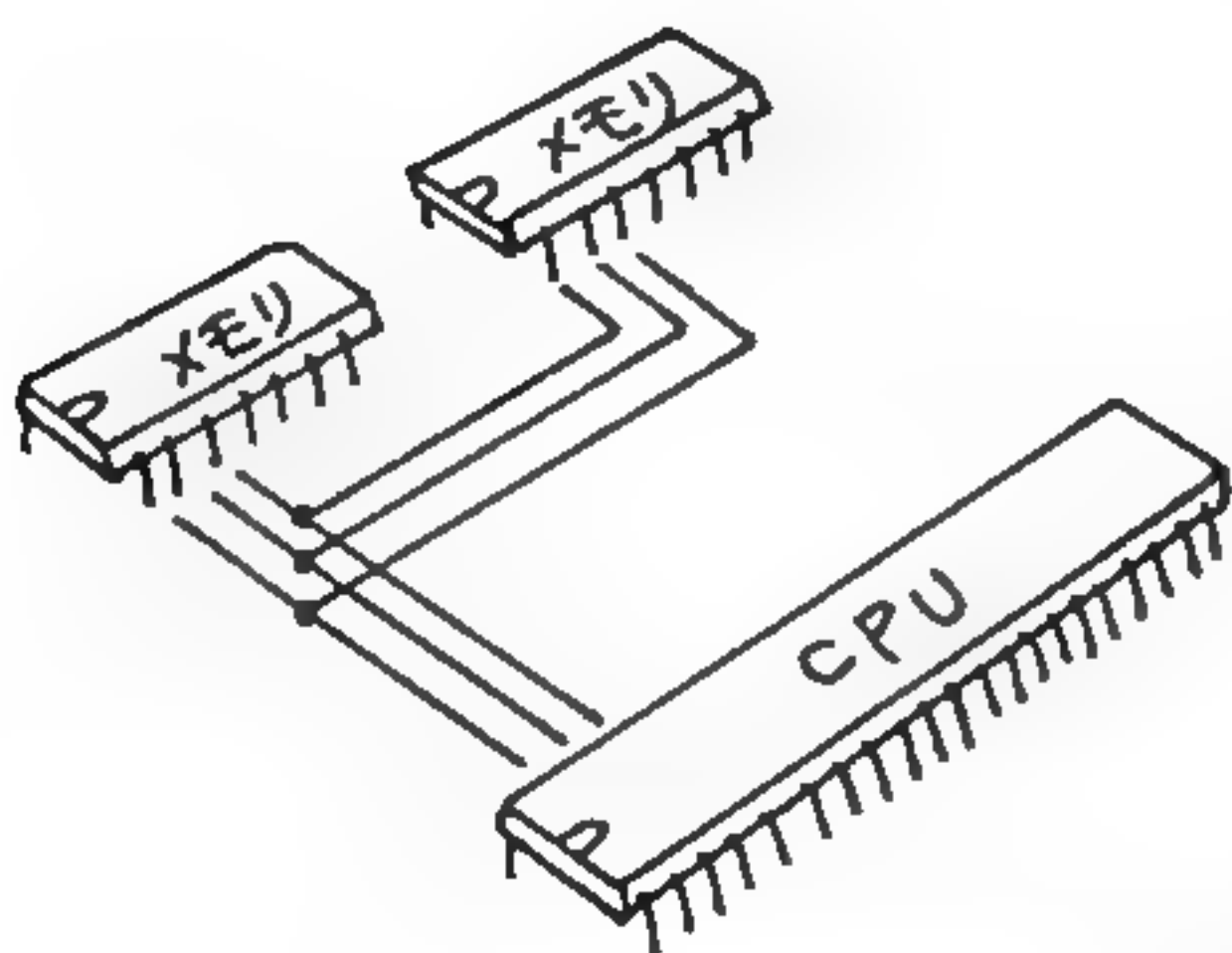
A2	A1	A0	選ばれるメモリ
0	0	0	0 番
0	0	1	1 番
0	1	0	2 番
0	1	1	3 番
1	0	0	4 番
1	0	1	5 番
1	1	0	6 番
1	1	1	7 番

この8バイトの中のどれかを指定するためには3ビットの信号があればいいんですね。1CのピンにはたぶんA0、A1、A2という3つのピンが出ているでしょう。



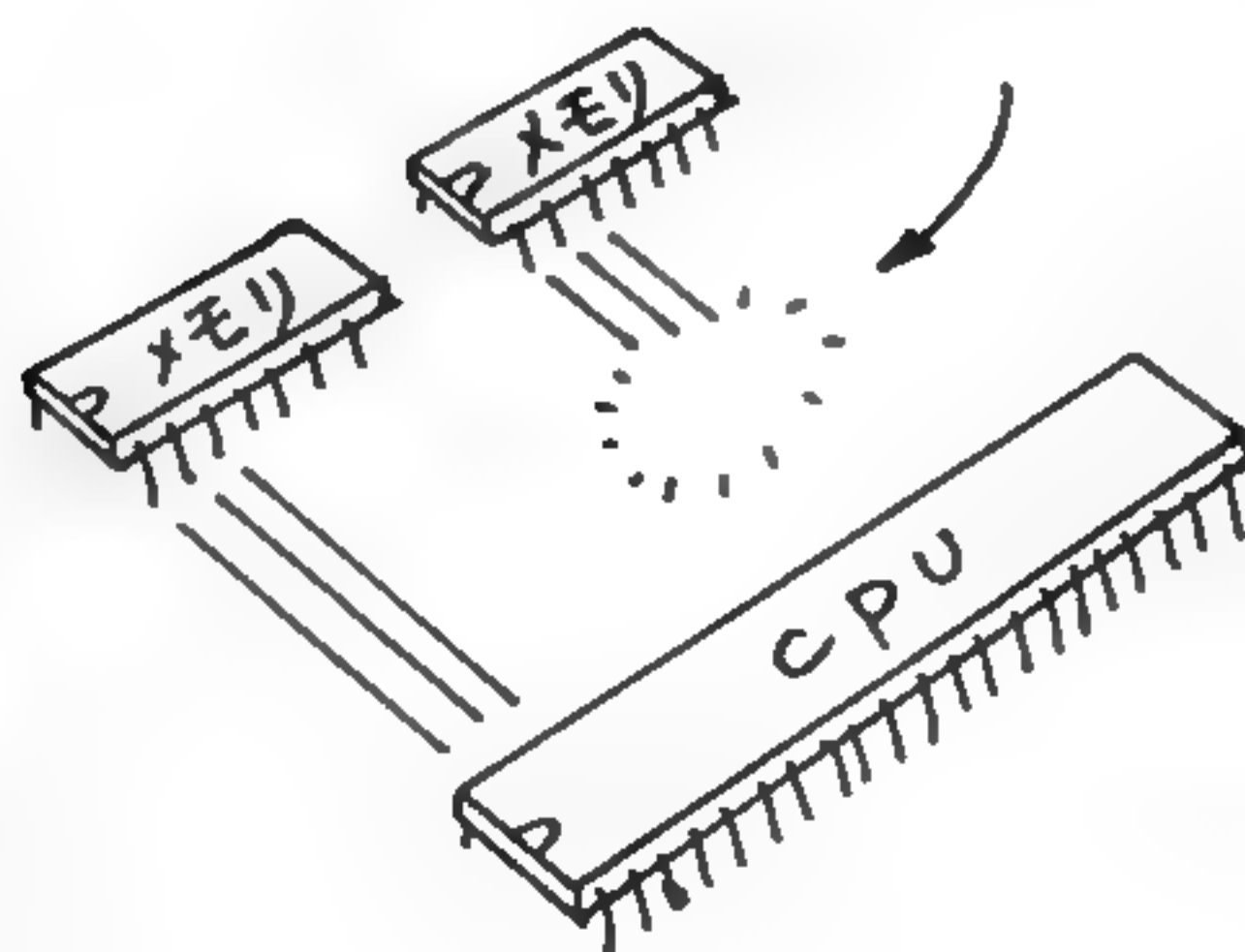
A2 A1 A0  
アドレス指定用ピン

結論からいいますと、下のようにつないだらいいのです。ただ2つのチップを使いわけ工夫が必要になります。

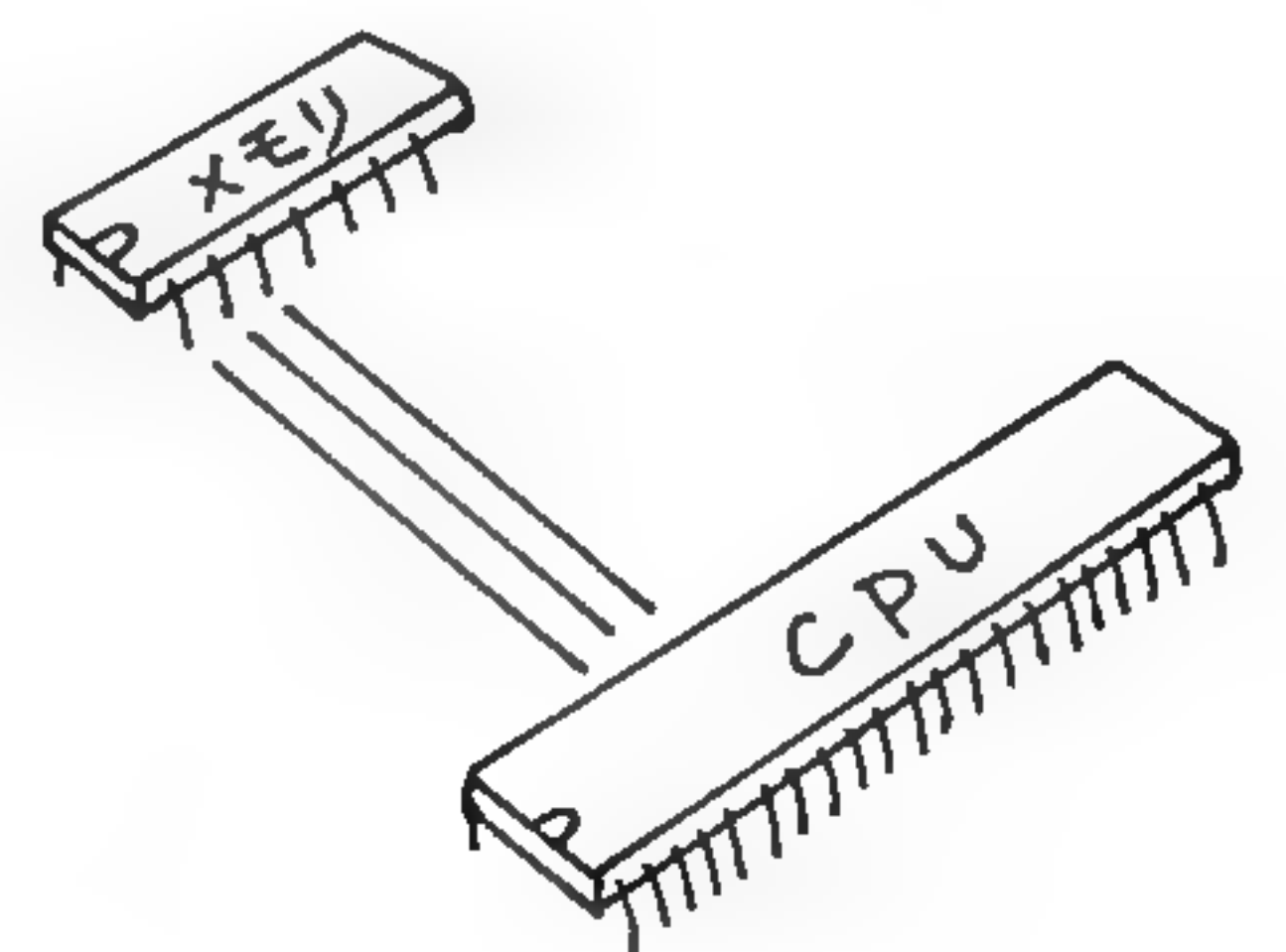


それでは同じメモリをもう1個使って0からFまで16バイトのメモリにしたい時はどうしたらよいのでしょうか？

どうつないだら  
よいか？

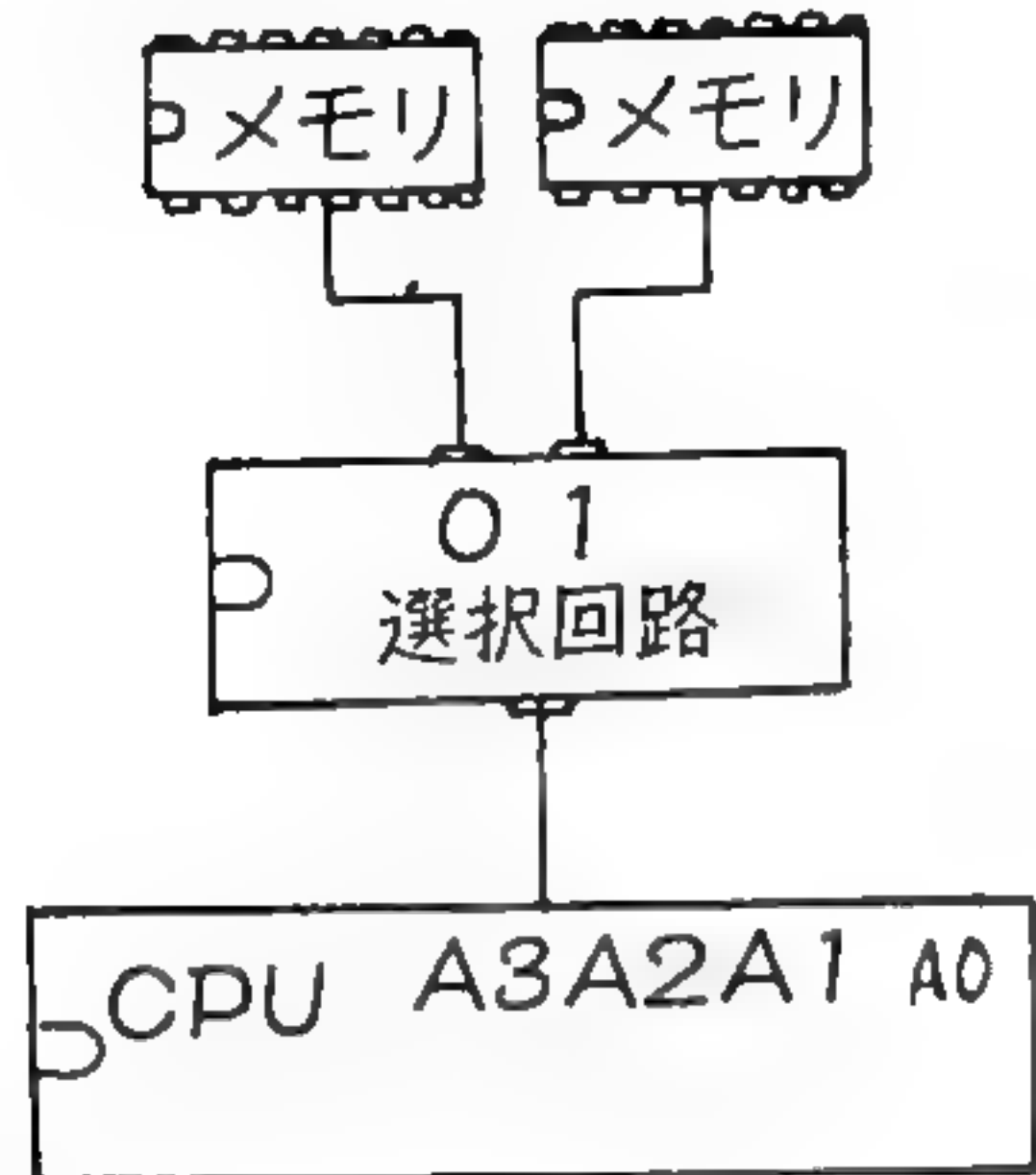


このアドレスピンをCPUのアドレスピンにつなげばCPUは0から7までの8バイトのメモリを持つことになります。

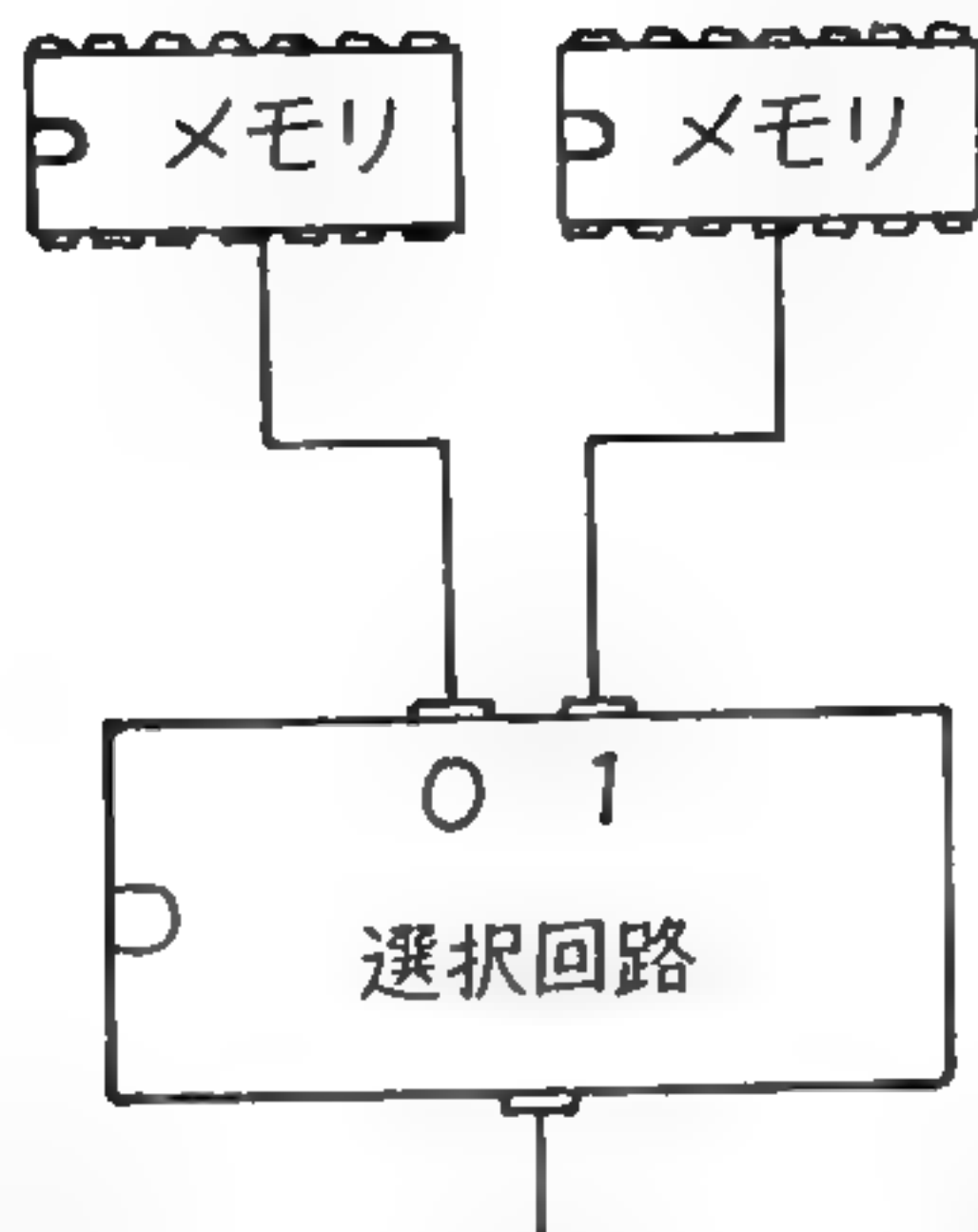




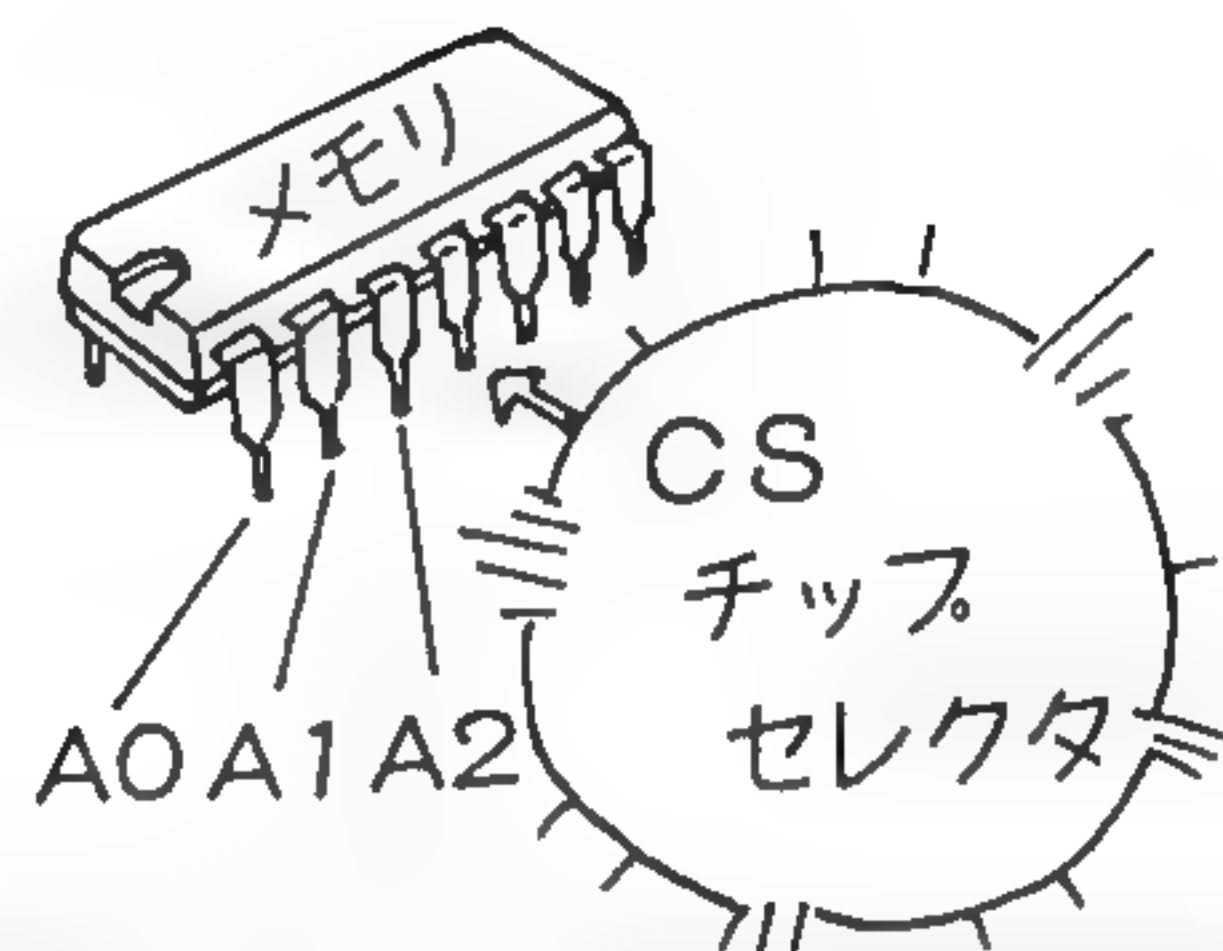
選択回路の先はCPUのもう1ビット上のA3につながります。これでうまくいくはずです。



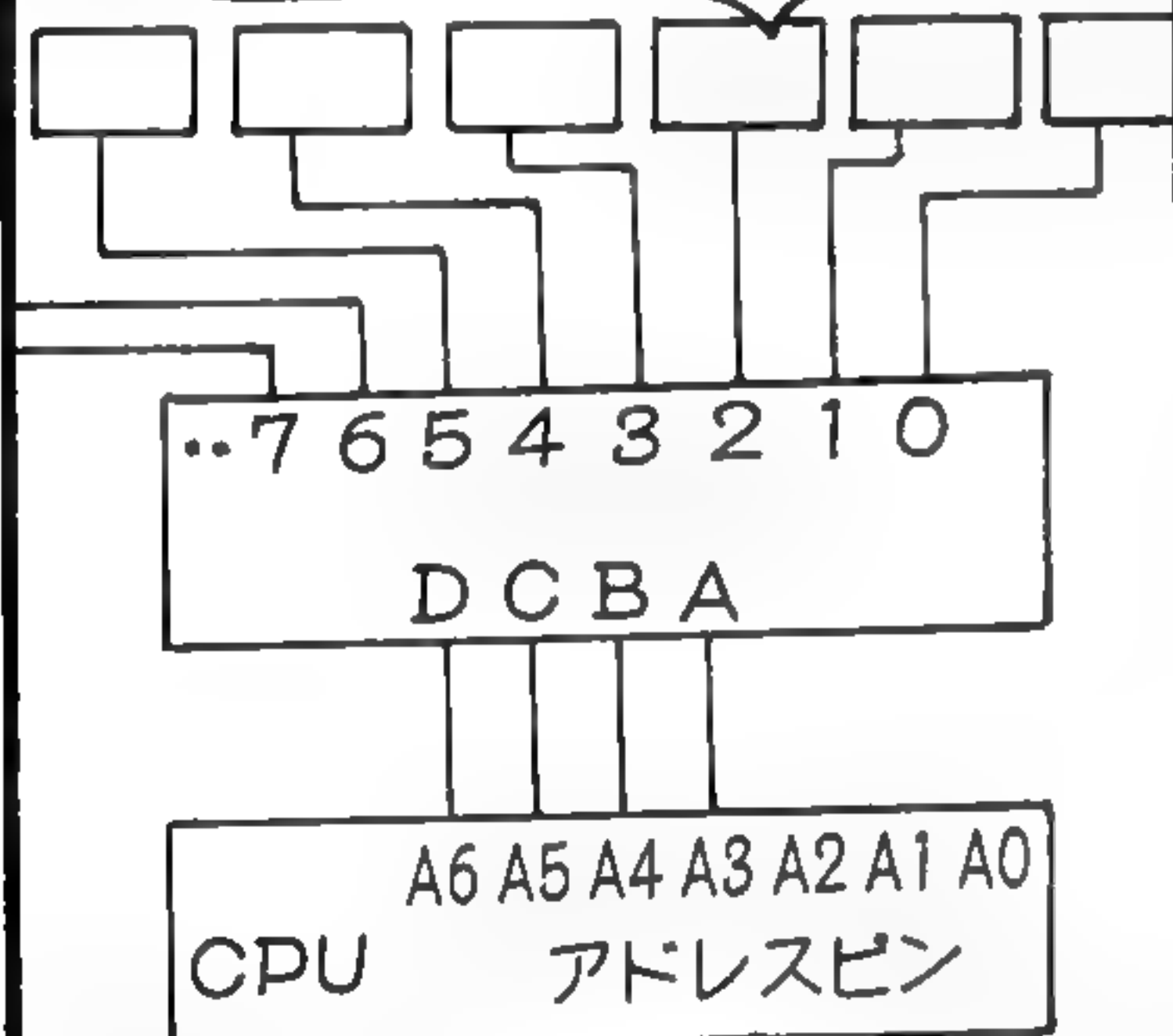
このチップセレクトピンで何番目のチップのメモリを使うのかを指定してやればいいのです。



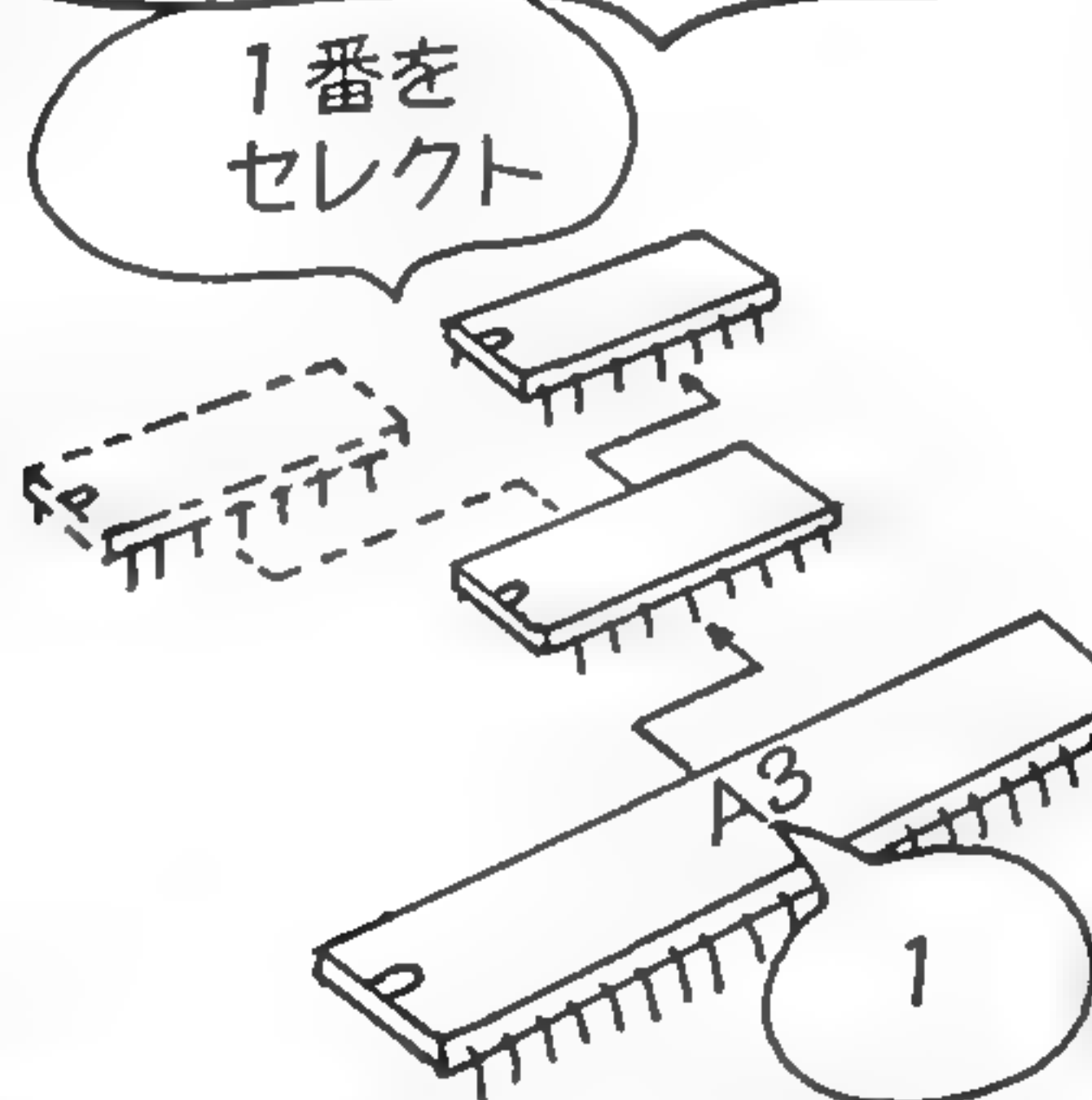
このチップの使い分けをチップセレクトと呼んでいます。そうです。内部メモリにアドレスを持つICチップにはこのチップセレクトピンがついているのです。



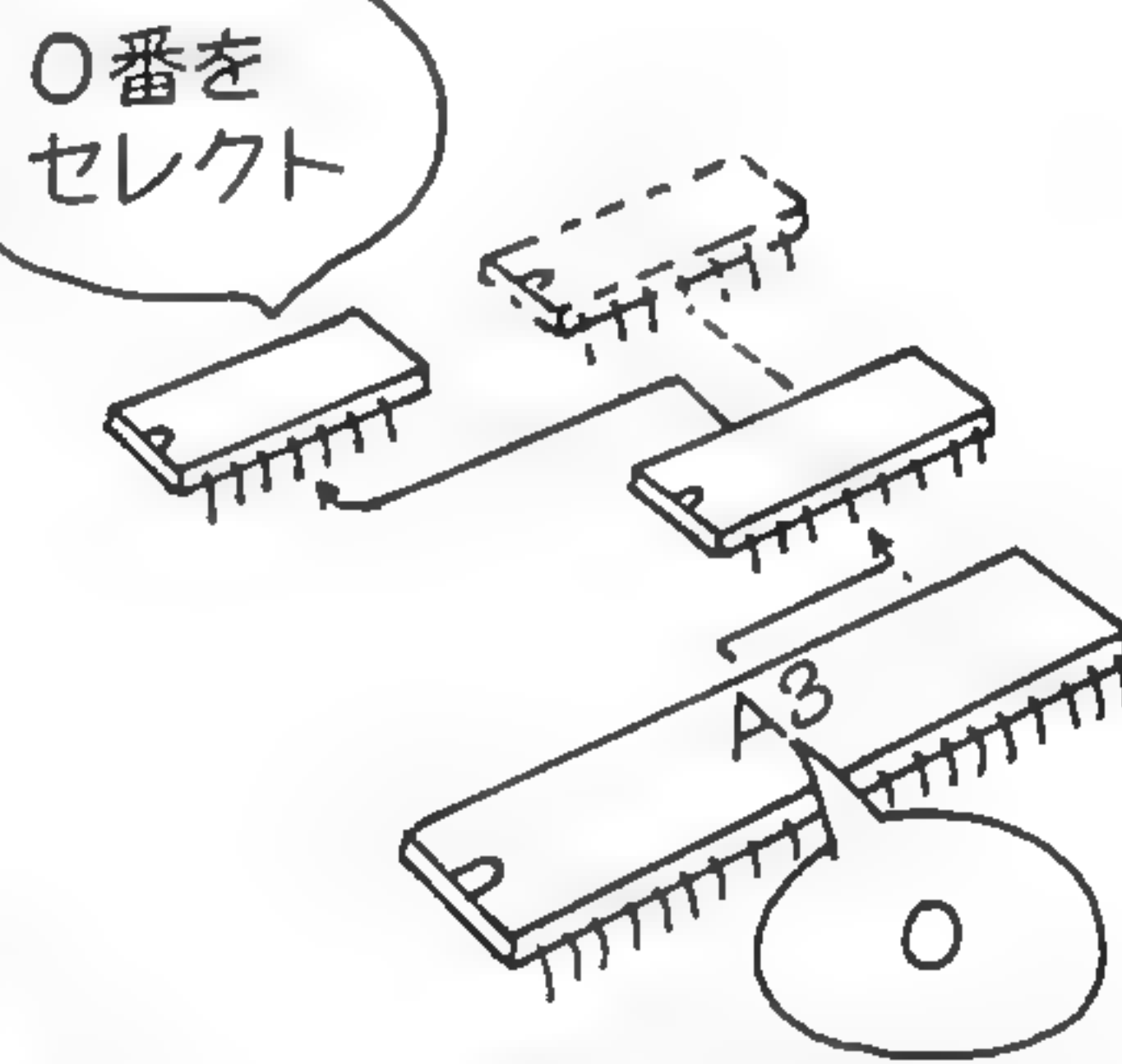
チップの数が増えたらA4、A5...と上位アドレスを選択回路につなげばよいわけですね。



次に8~Fのアドレスが出力された場合はA3は1ですから1番のチップが選ばれるというしかけです。



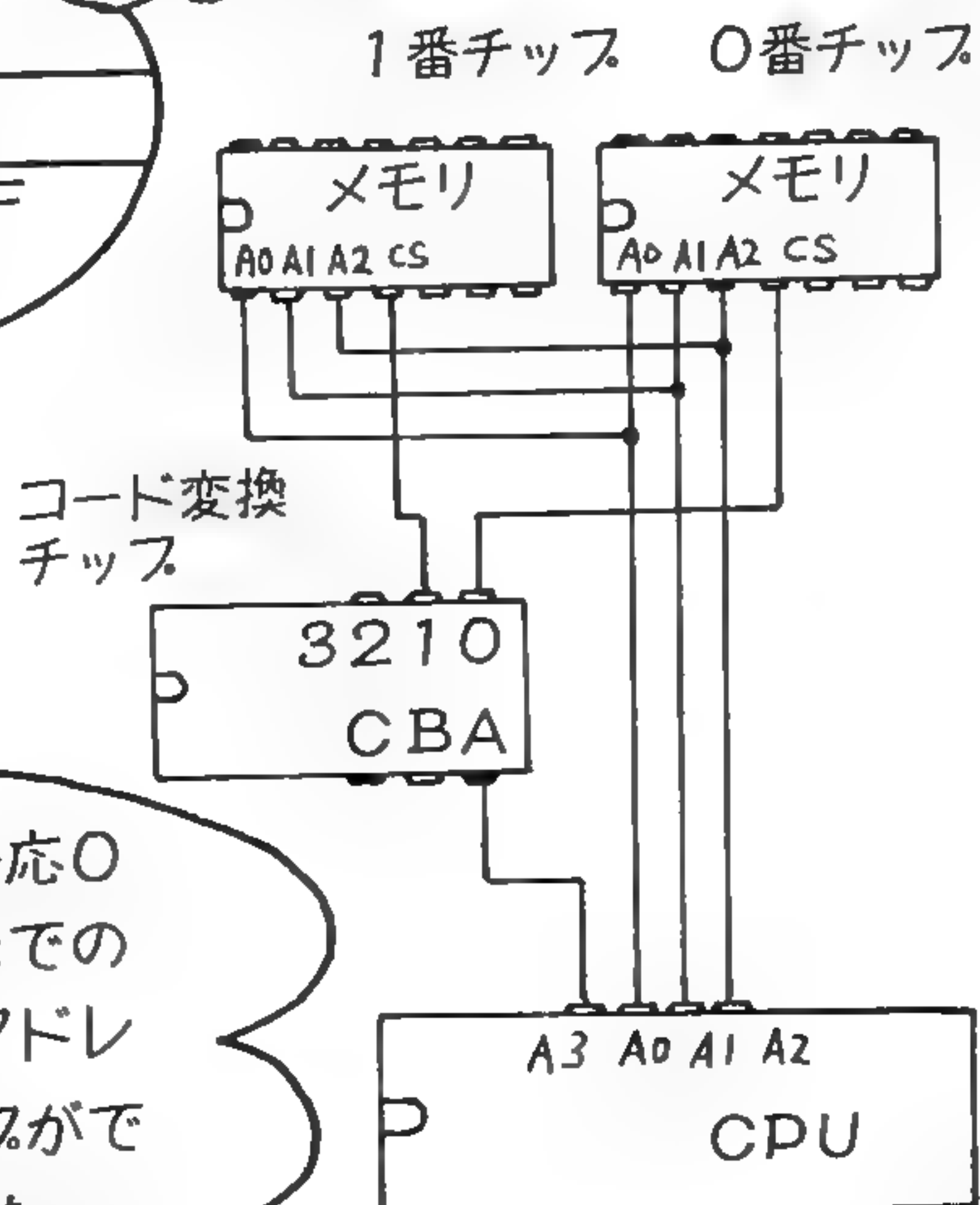
CPUから出るアドレス信号で0~7の場合はA3は0ですから0番のチップが選ばれています。



0番 8バイト 1番 8バイト  
0000 0007 000F  
メモリマップ



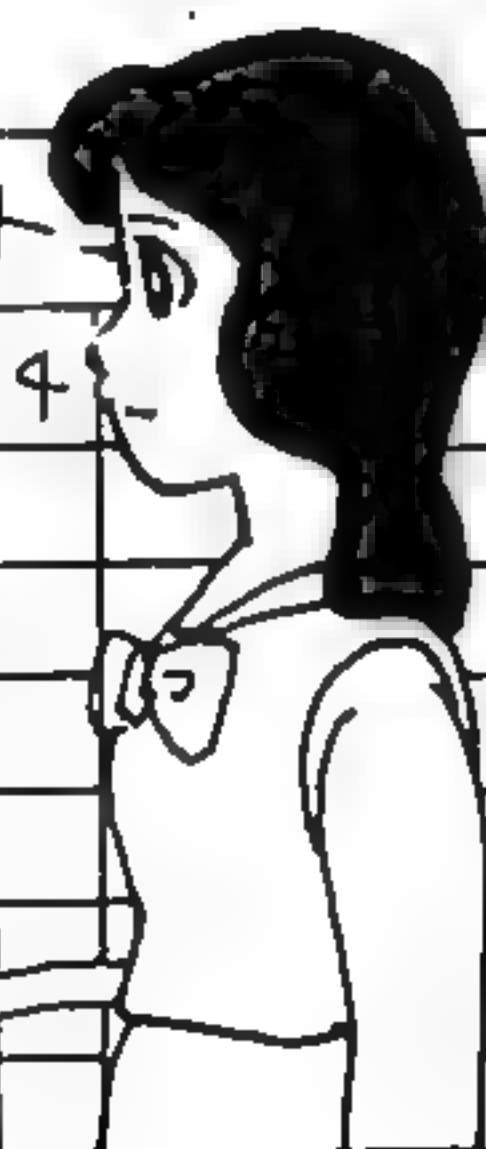
これで一応0からFまでのメモリアドレスマップができました。



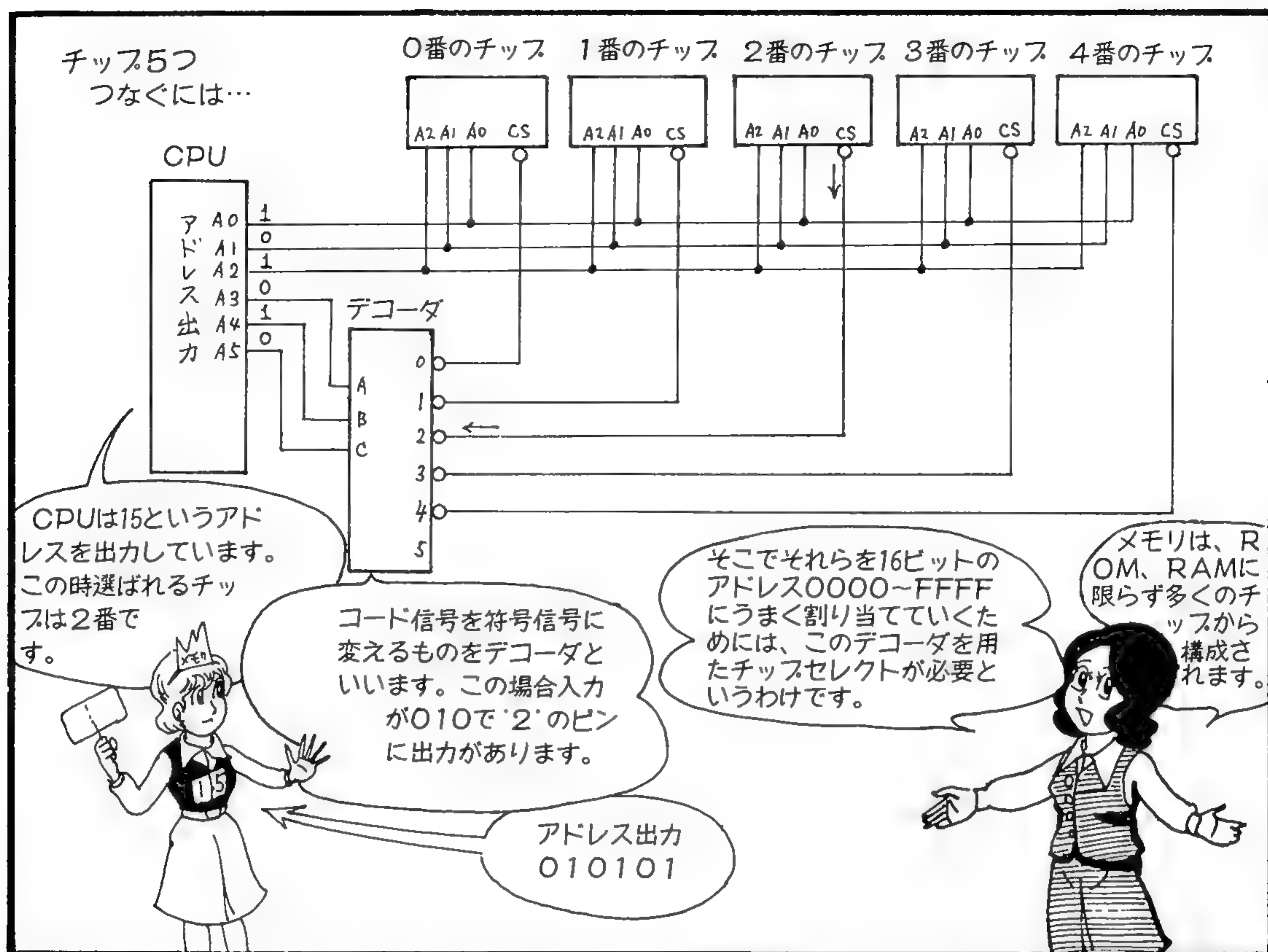
この選択回路は入力された2進化コードを10進化して出力するものでいいですね。このような機能をもったICもちゃんと（マイコンより昔から）売られているのです。

2進→10進変換IC

インプット				アウトプット			
D	C	B	A	0	1	2	3 4
0	0	0	0	L			
0	0	0	1		L		
0	0	1	0			L	
0	0	1	1				L
1	1	1	1				

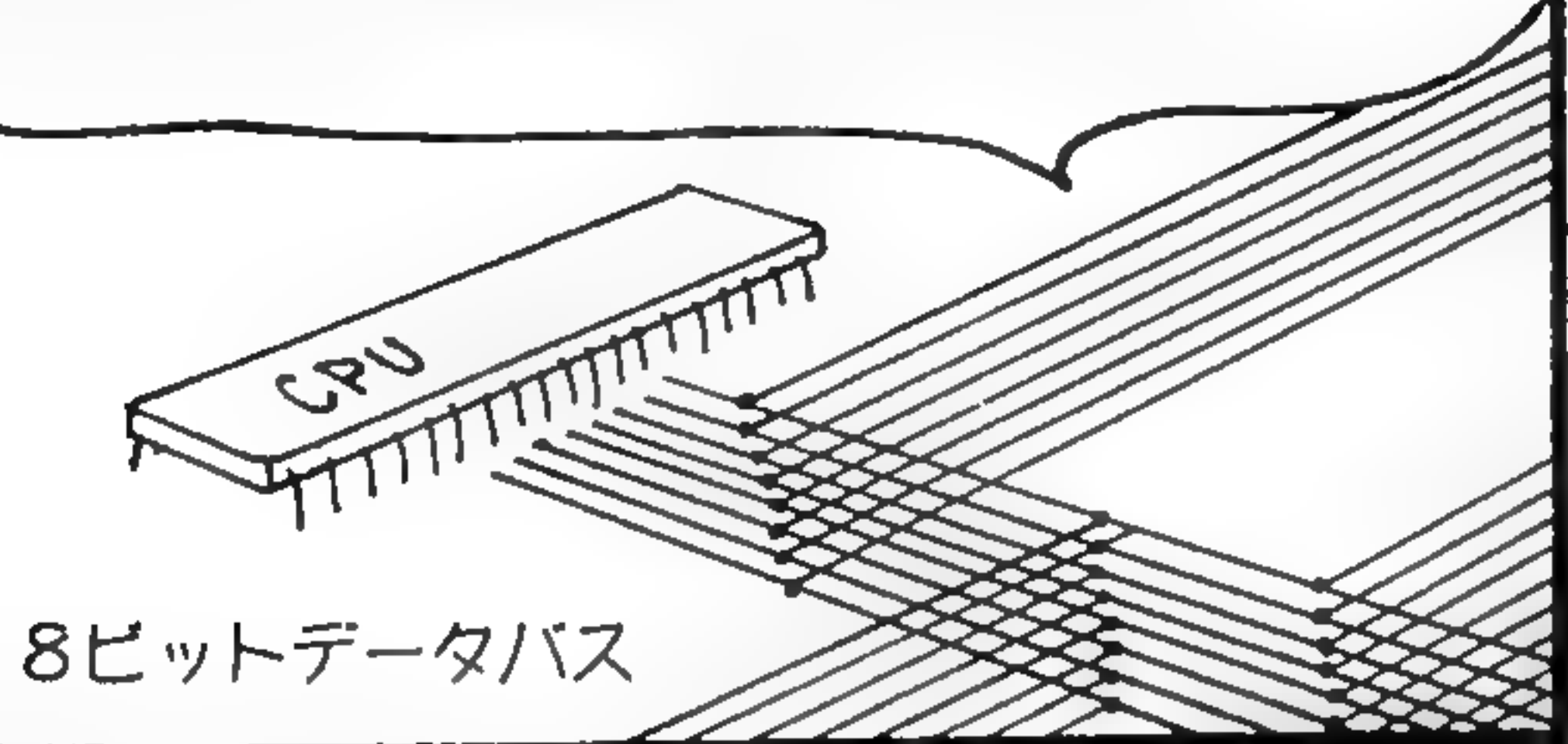








メモリアポートのデータ線はすべてCPUにつながっています。CPUのデータピンが8つですんでいるのはそれぞれのアドレスのデータがこの8つのピンを共有しているからです。



## ⑦ トライステート・ゲート

第3の状態はハイ・インピーダンスです。

HiZ

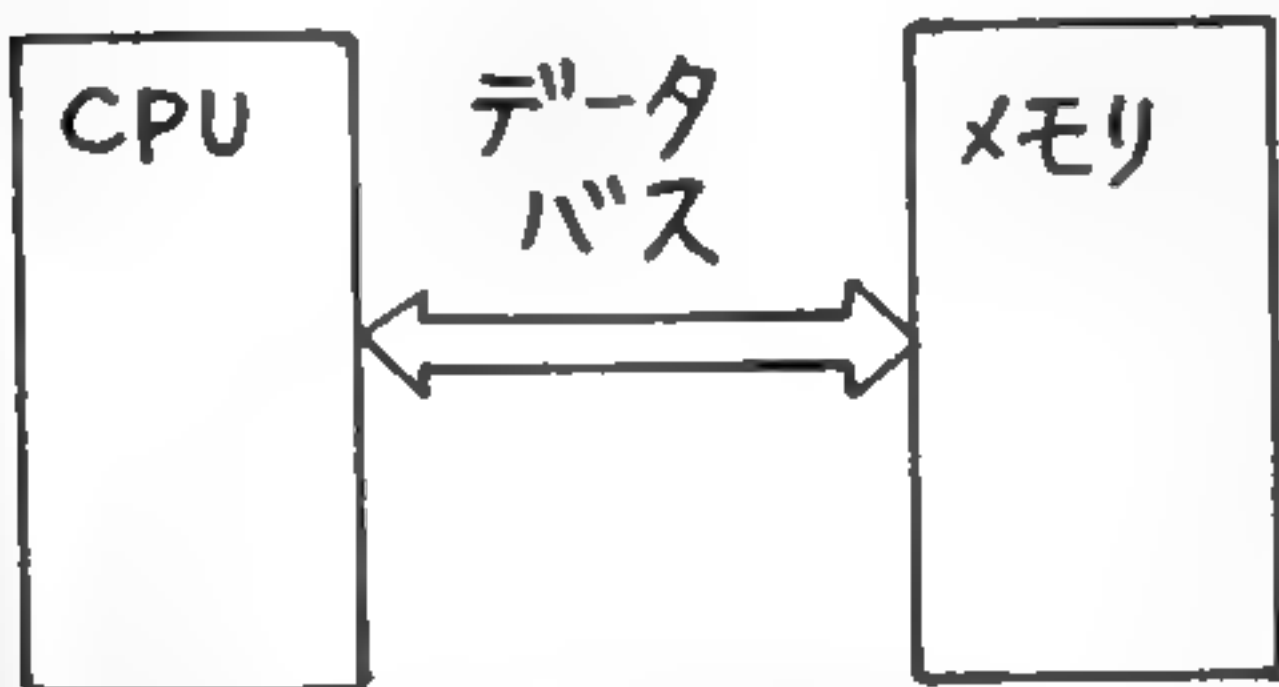
1 (ハイ)  
0 (ロー)  
そして  
.....

このあたりはCPU制御信号とも関係があります。じっくり読んでいってください。

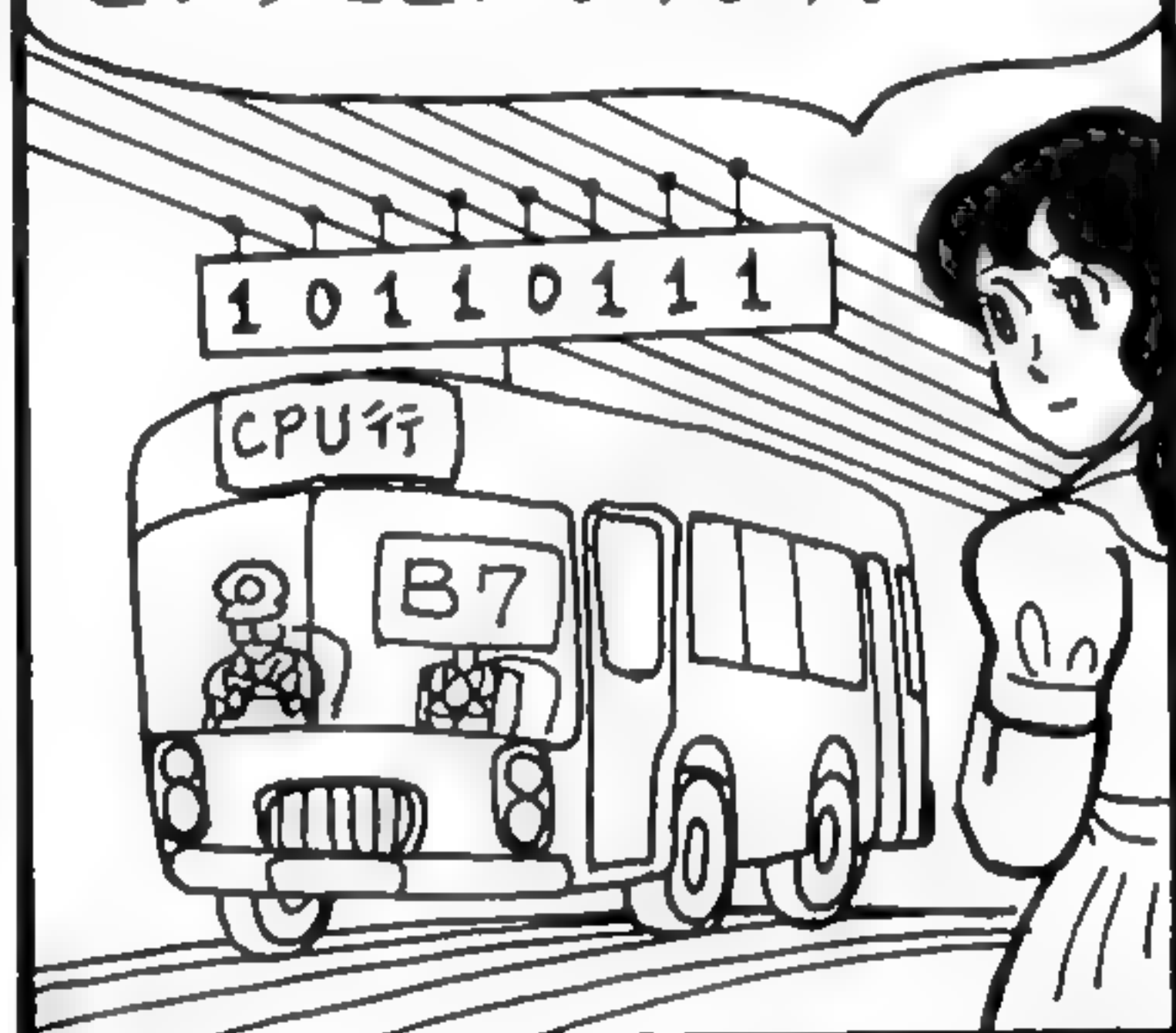
だんだん  
むずかしく  
なってくる  
わ...

HiZ  
ハイインピー  
ダンス  
フローティング  
浮いている

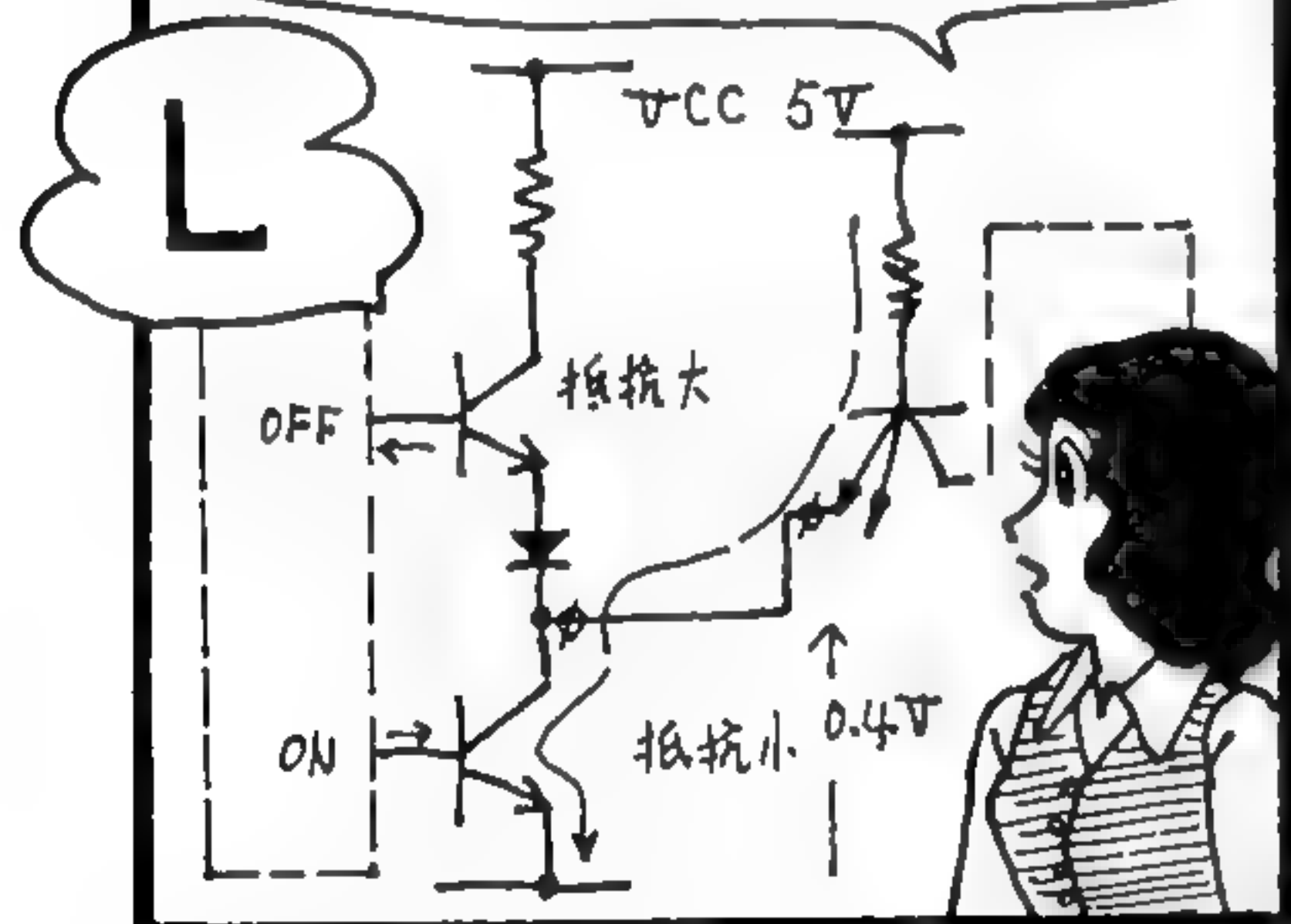
この矢印はデータの方を示していますが、ある時間には←か→のどちらかにしか動きません。



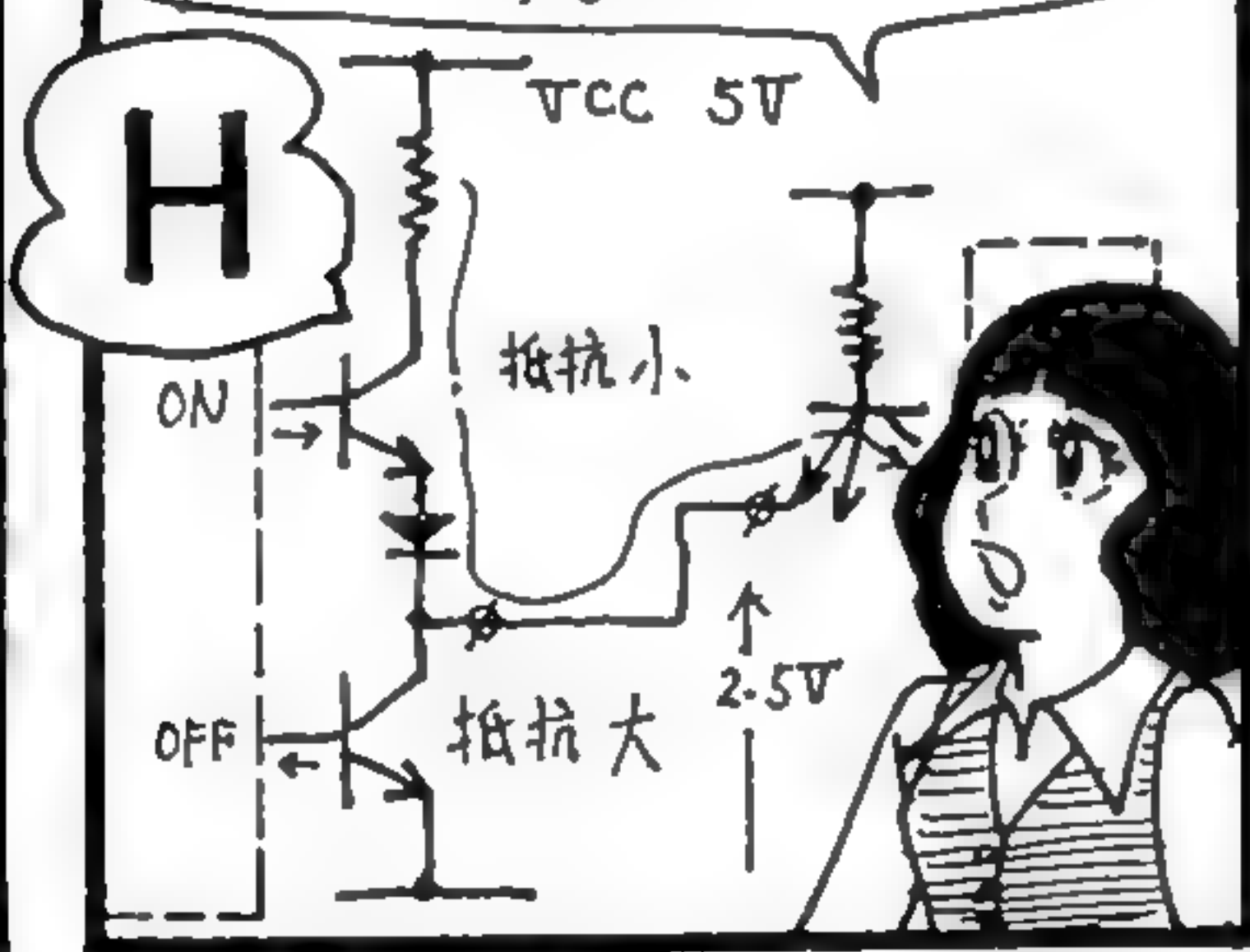
このデータピンの共用がバスということになります。



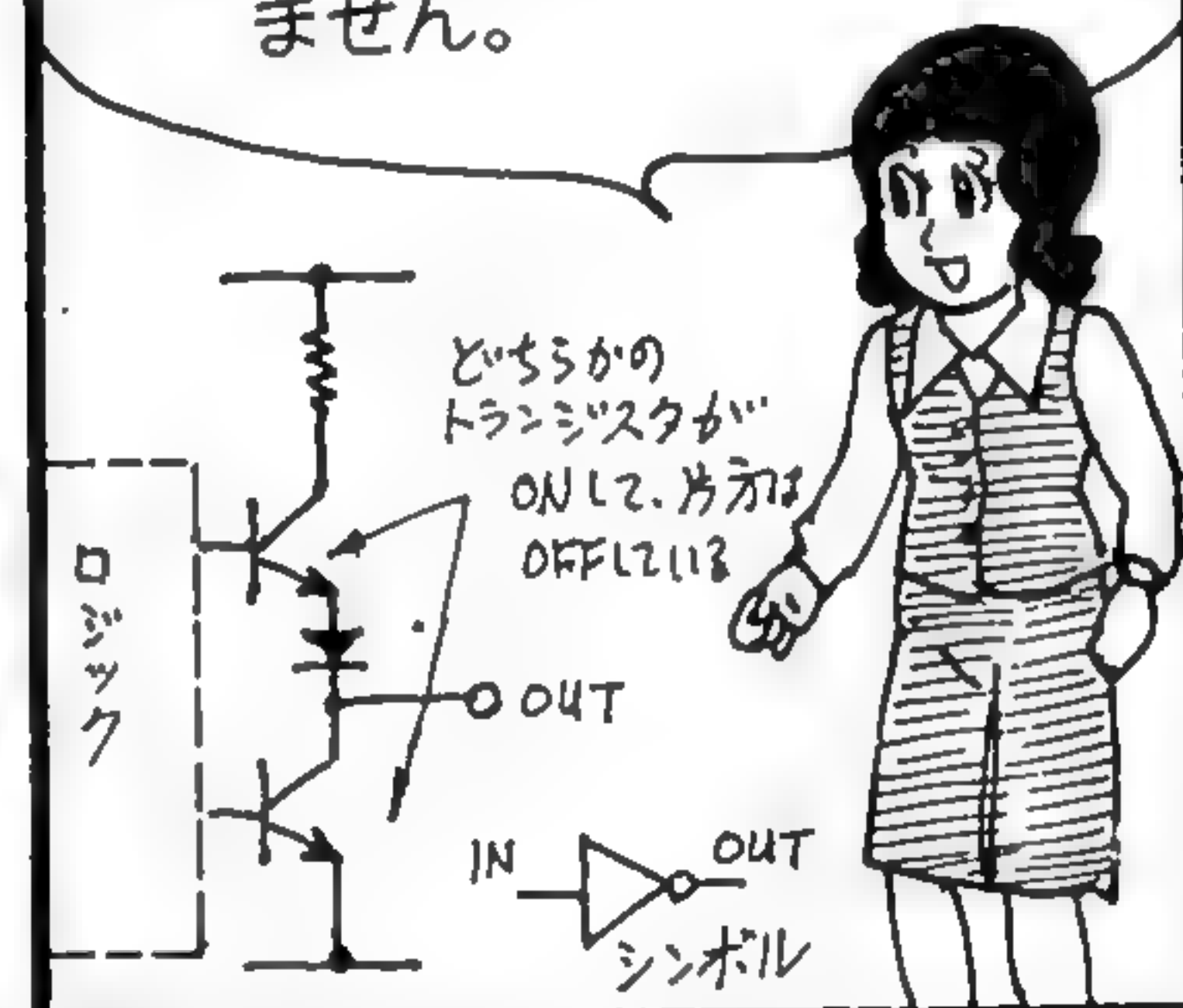
下のトランジスタがONしている時は上のトランジスタはOFFとなり'L'を出力します。



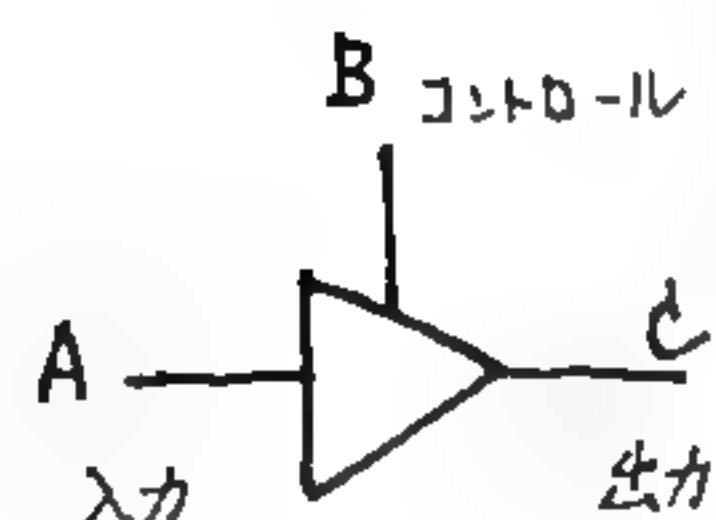
上のトランジスタがONしている時は下のトランジスタはOFFとなり'H'を出力します。



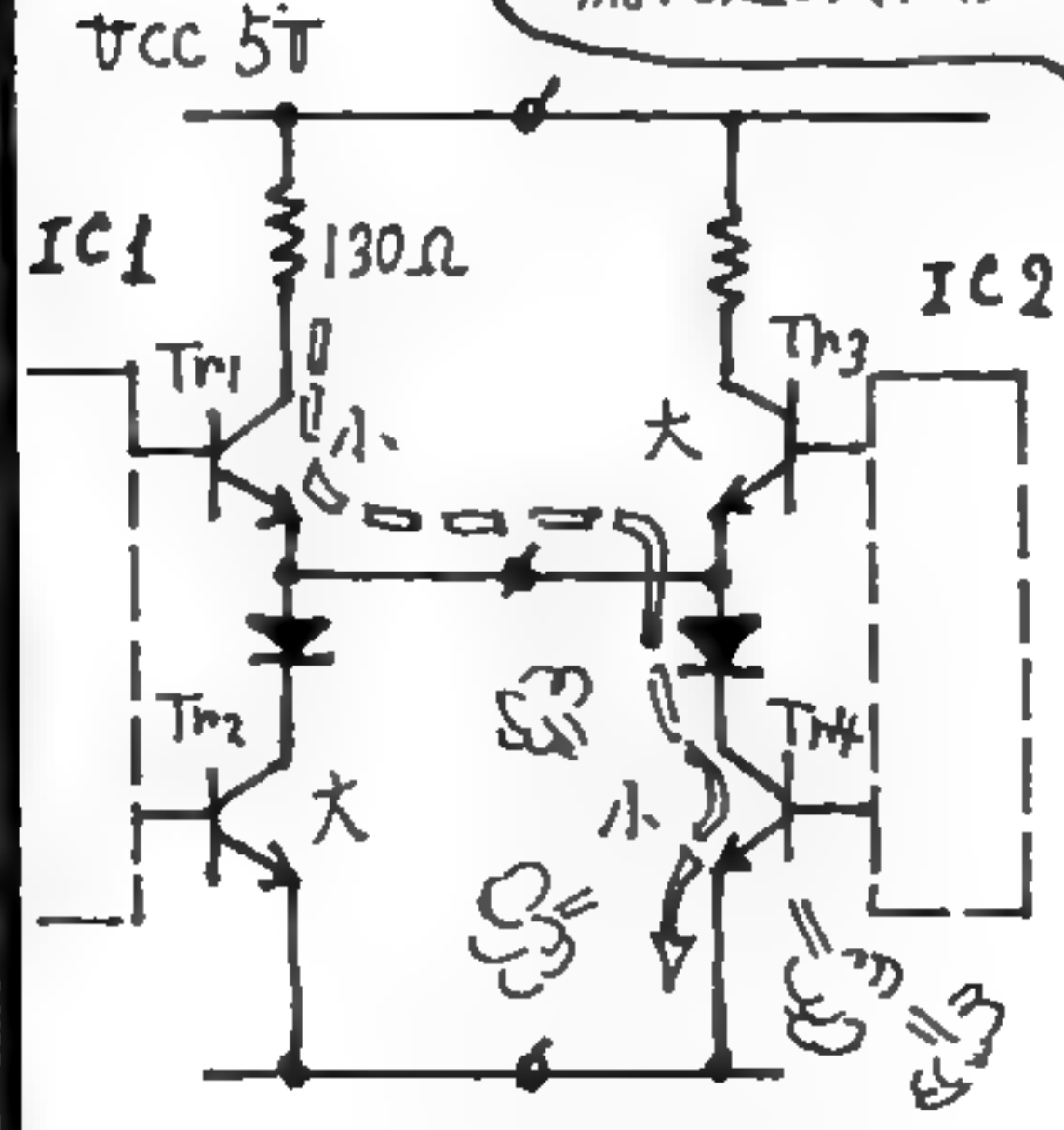
普通のゲートはHかLのどちらかの出力以外の状態はとれません。



そこで上下のトランジスタを両方同時にOFFにするコントロール入力を設けたトライステートのゲートがあるのです。

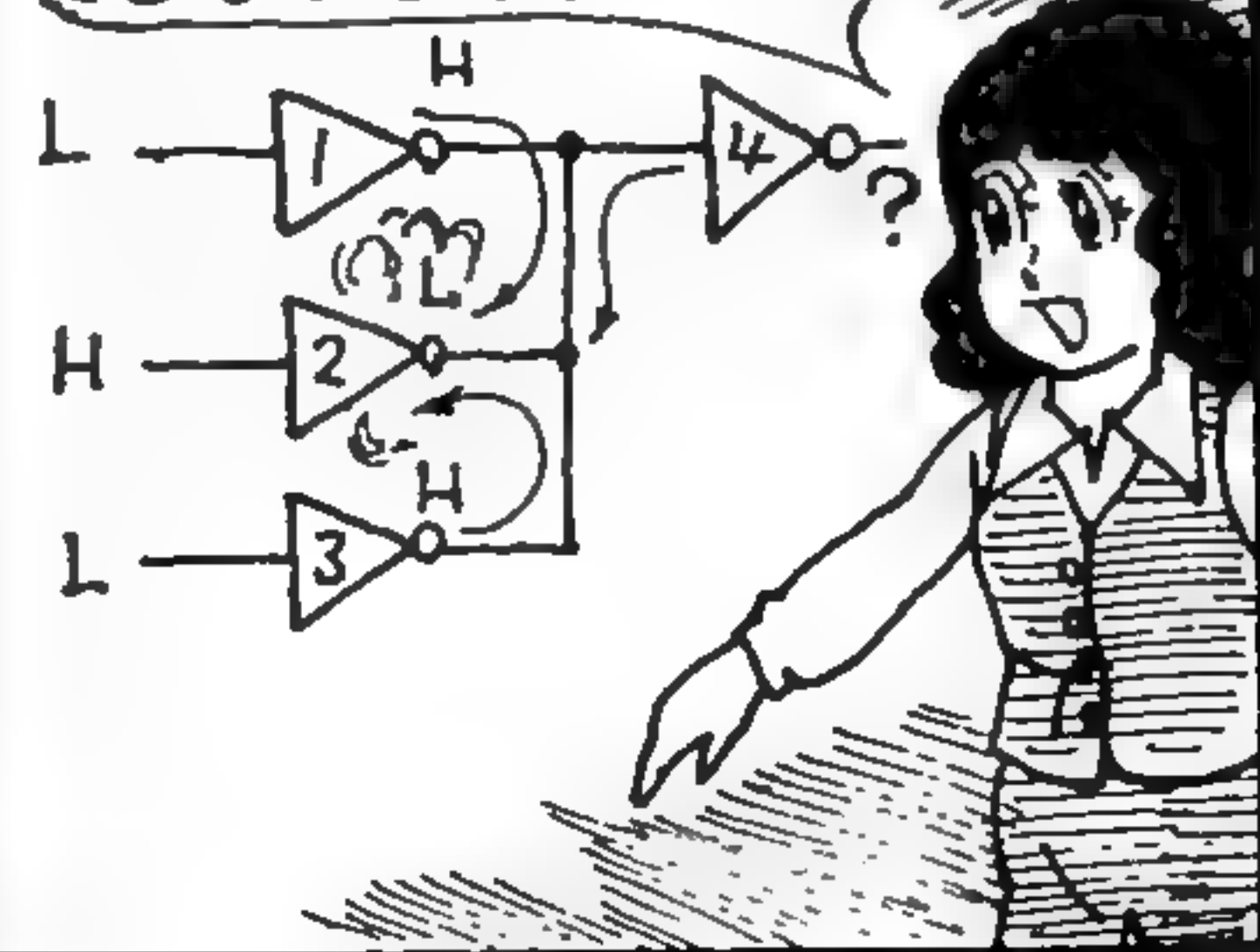


IC2のTr4に電流が流れ込み、パンクする。



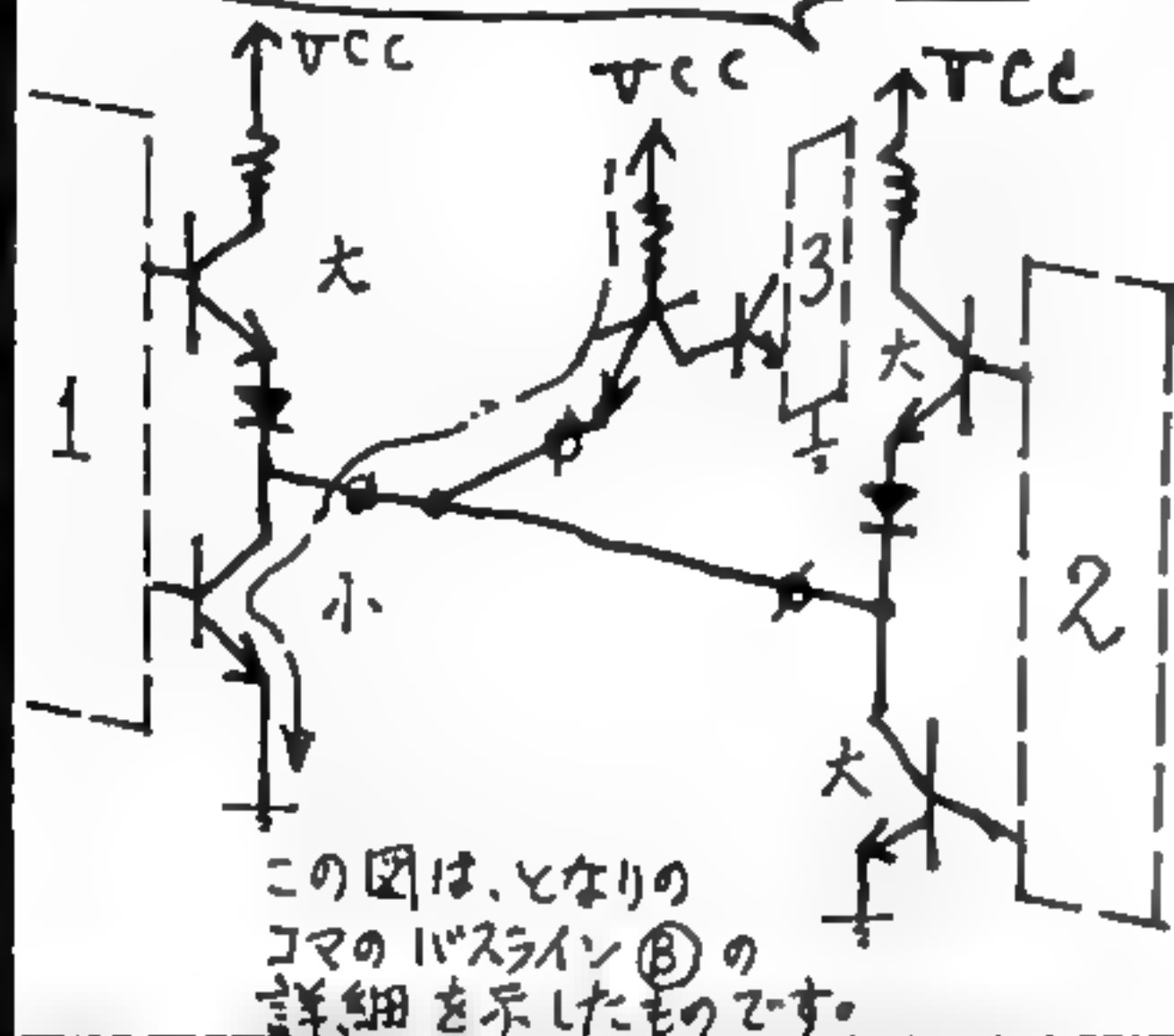
右のDの2つの出力部分です。

このようなトードムボールの出力は並列に接続することができません。このような論理になっている時は、Dのゲートが焼けてしまいます。

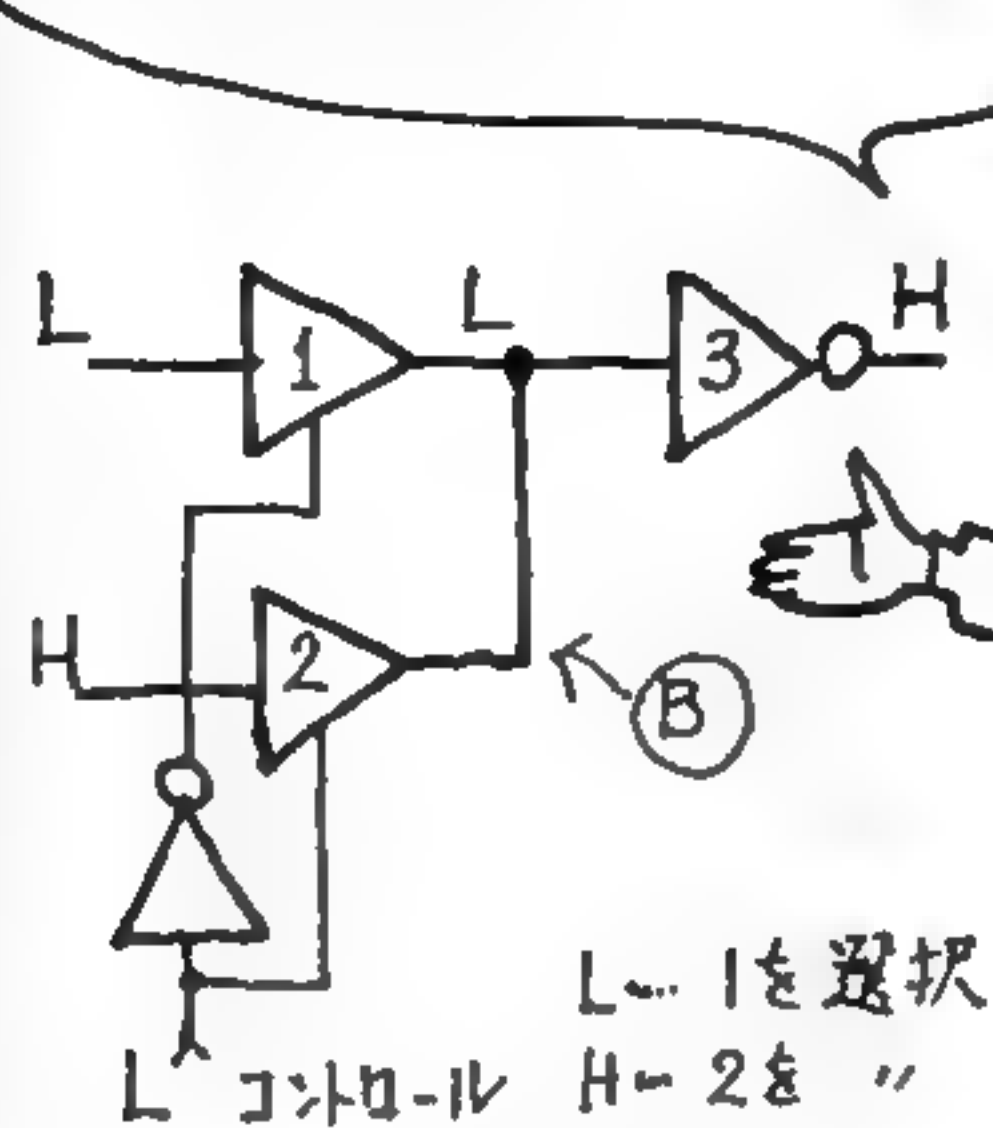




この場合IC2の出力はHiZとなり回路から切り離された状態です。この状態をフローティングというのです。

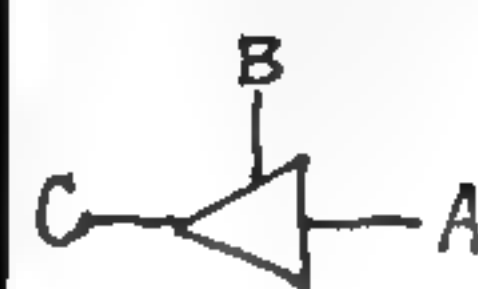


このように回路を組めば、出力同士のケンカはありません。だからバスラインとして使えるのです。

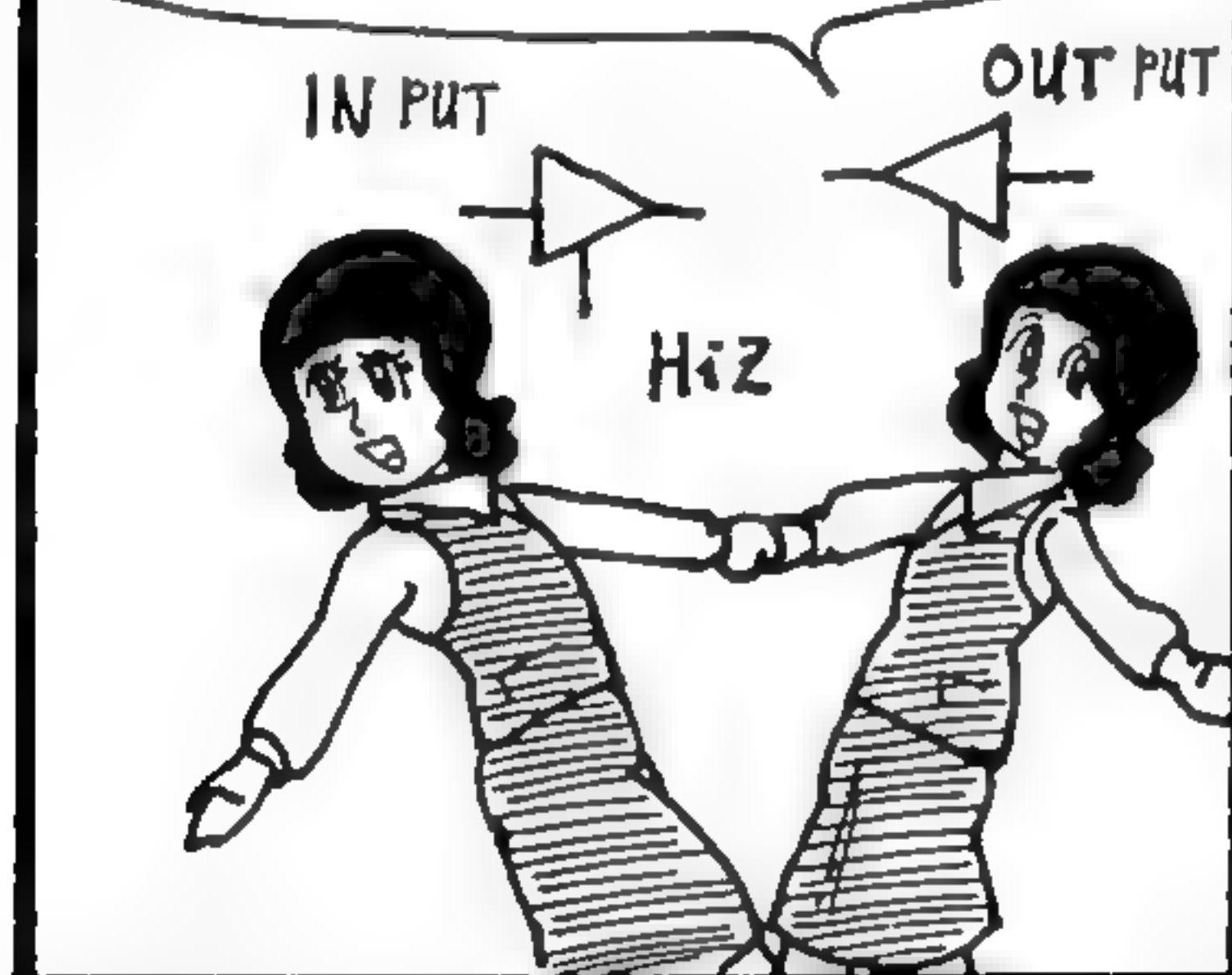


これが、トリステート・ゲートの真理値表です。HiZはハイ・インピーダンスのこと。上下のトランジスタがOFFの状態です。

A	B	C
0	0	HiZ
1	0	HiZ
0	1	0
1	1	1



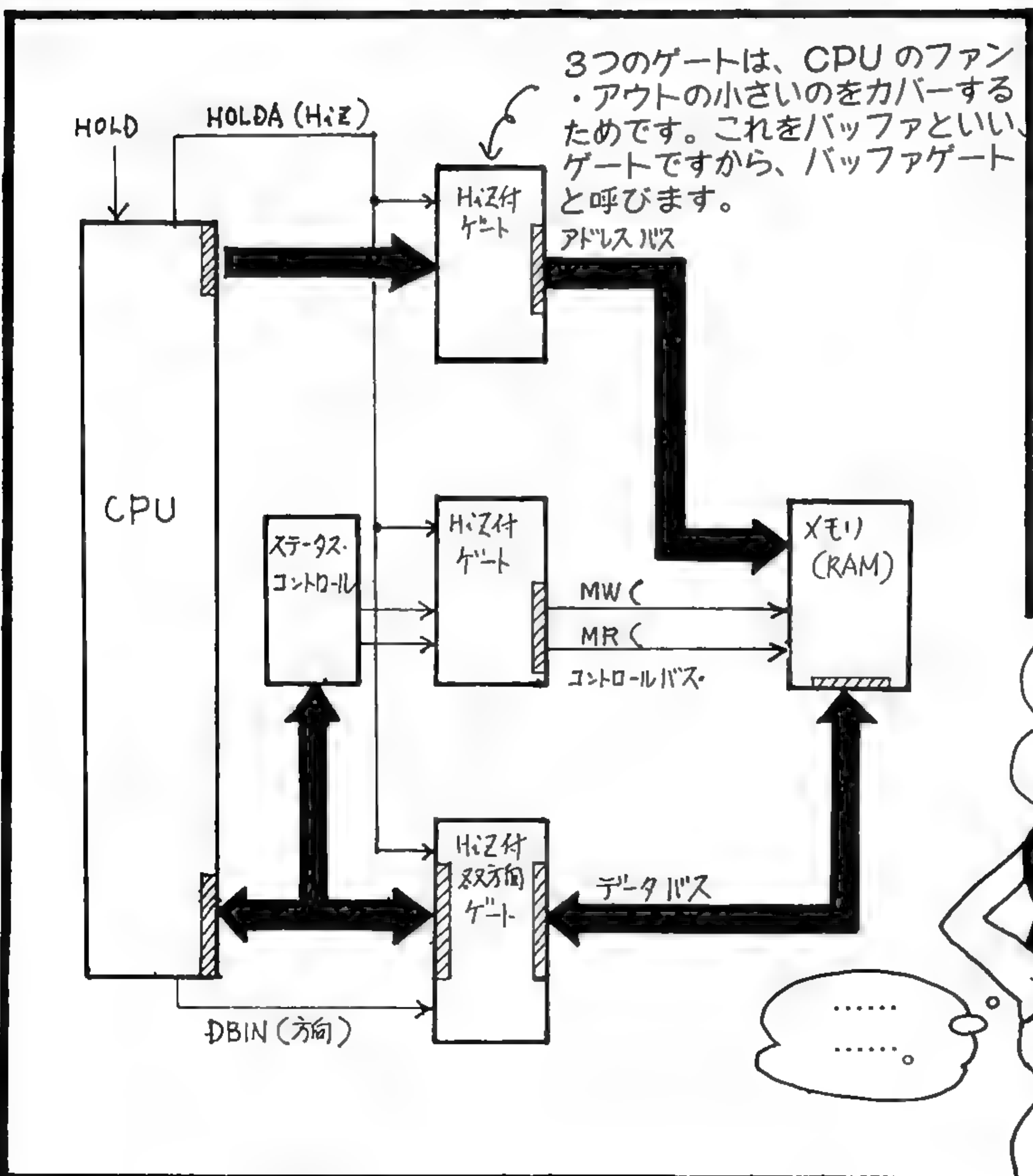
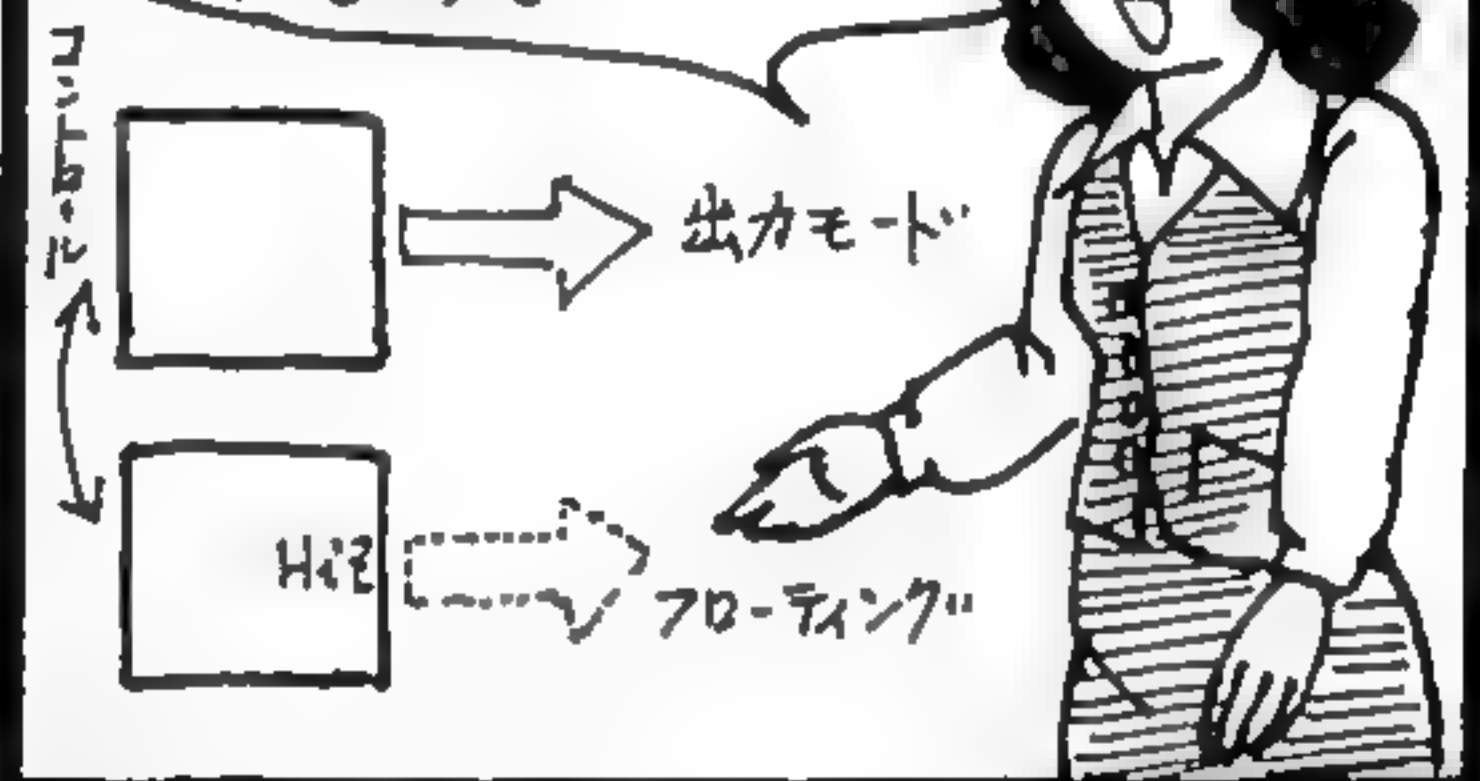
データバスの双方向性バスもこのトリステート・ゲートを応用しているのです。



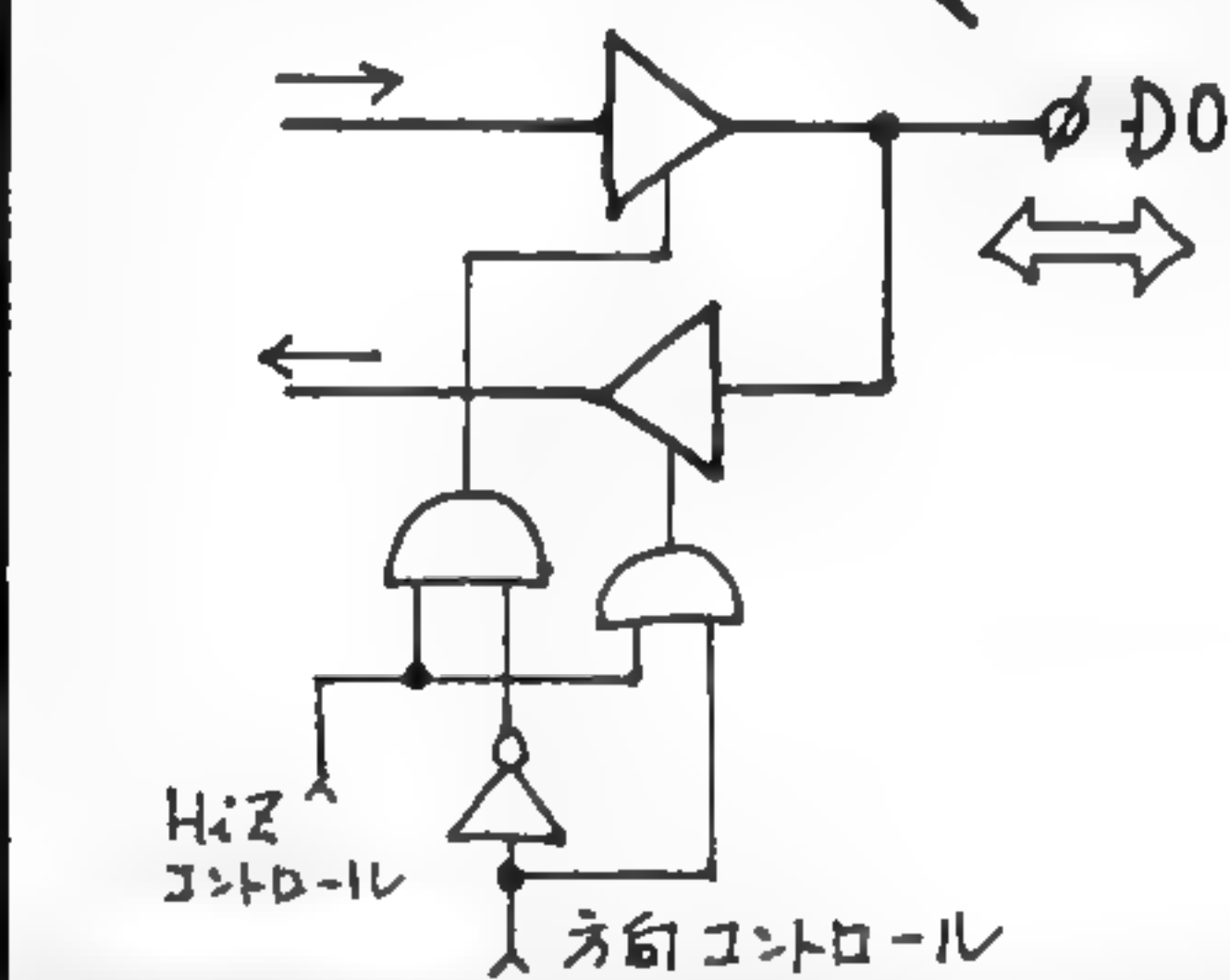
この回路はTTLのもんですがCPUやメモリはC-MOSです。でも、理屈は同じですから…。知りたければ、自分で勉強してください。私、ちょっと忙しいの。



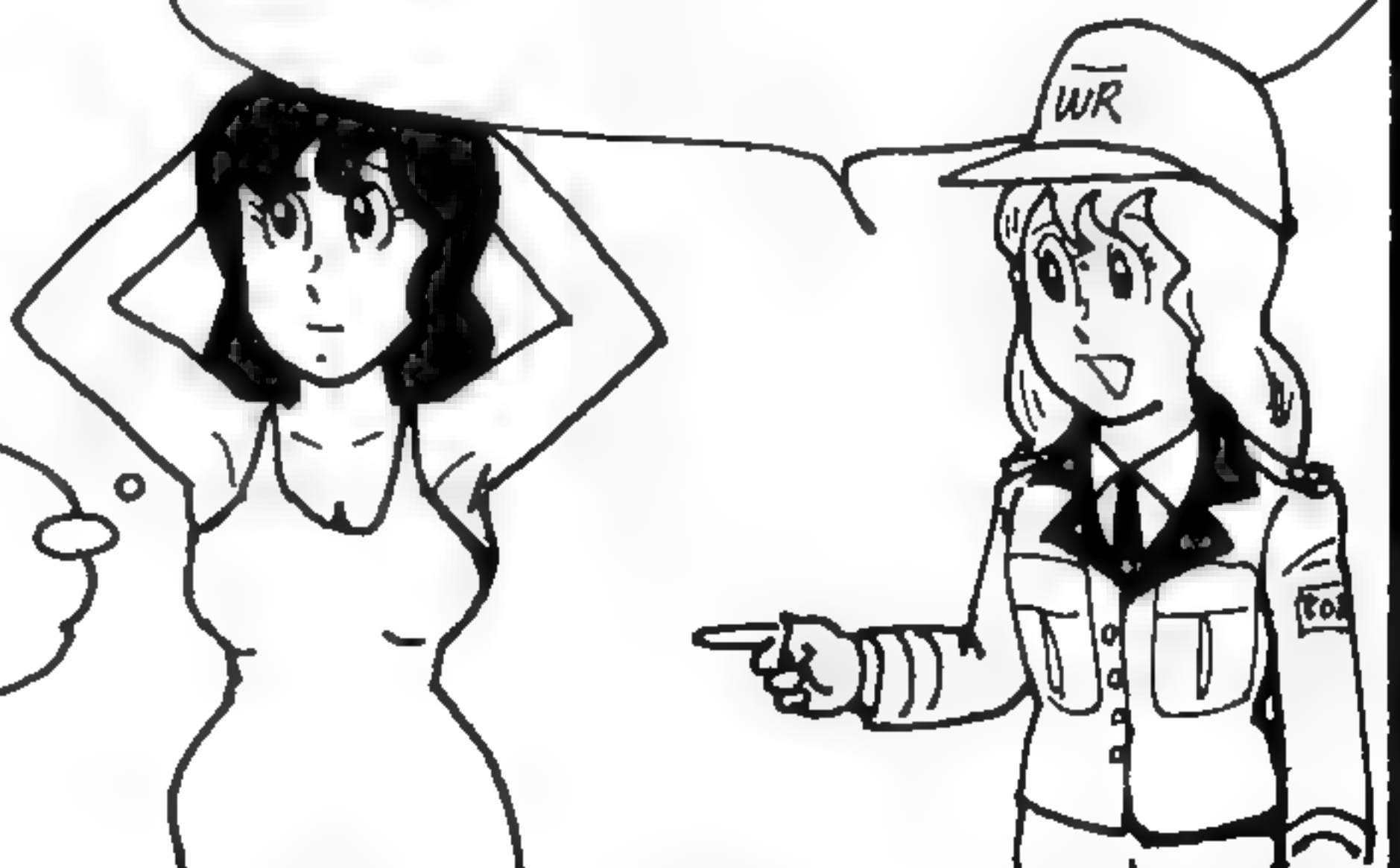
バスラインに接続されているチップで出力端子側は、すべてこのトリステートになっています。そうでないと、出力同士のケンカが起こって混乱してしまいます。



このようにトリステート・ゲートを組み合わせたら端子DOはコントロール信号によって双方向性になりますね？



図中□が、ステータスの出力部です。ですから選ばれていないアドレスの出力部はすべてバスに対して浮いています。





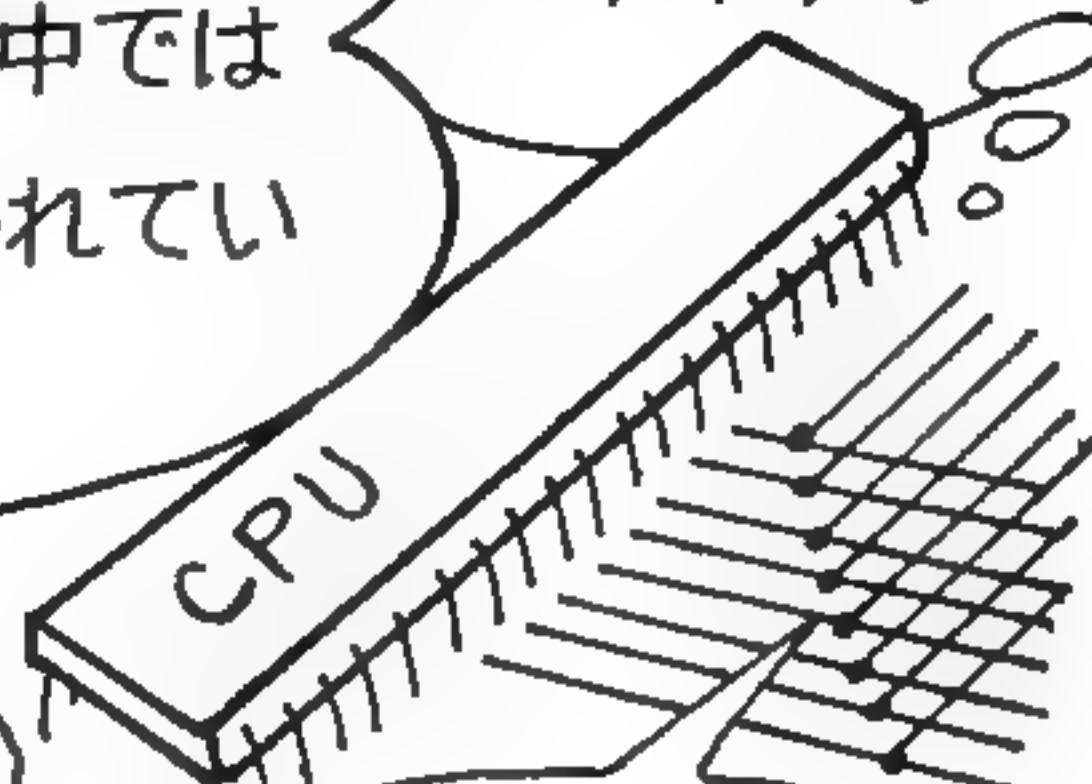


## ⑧ CPUと制御信号



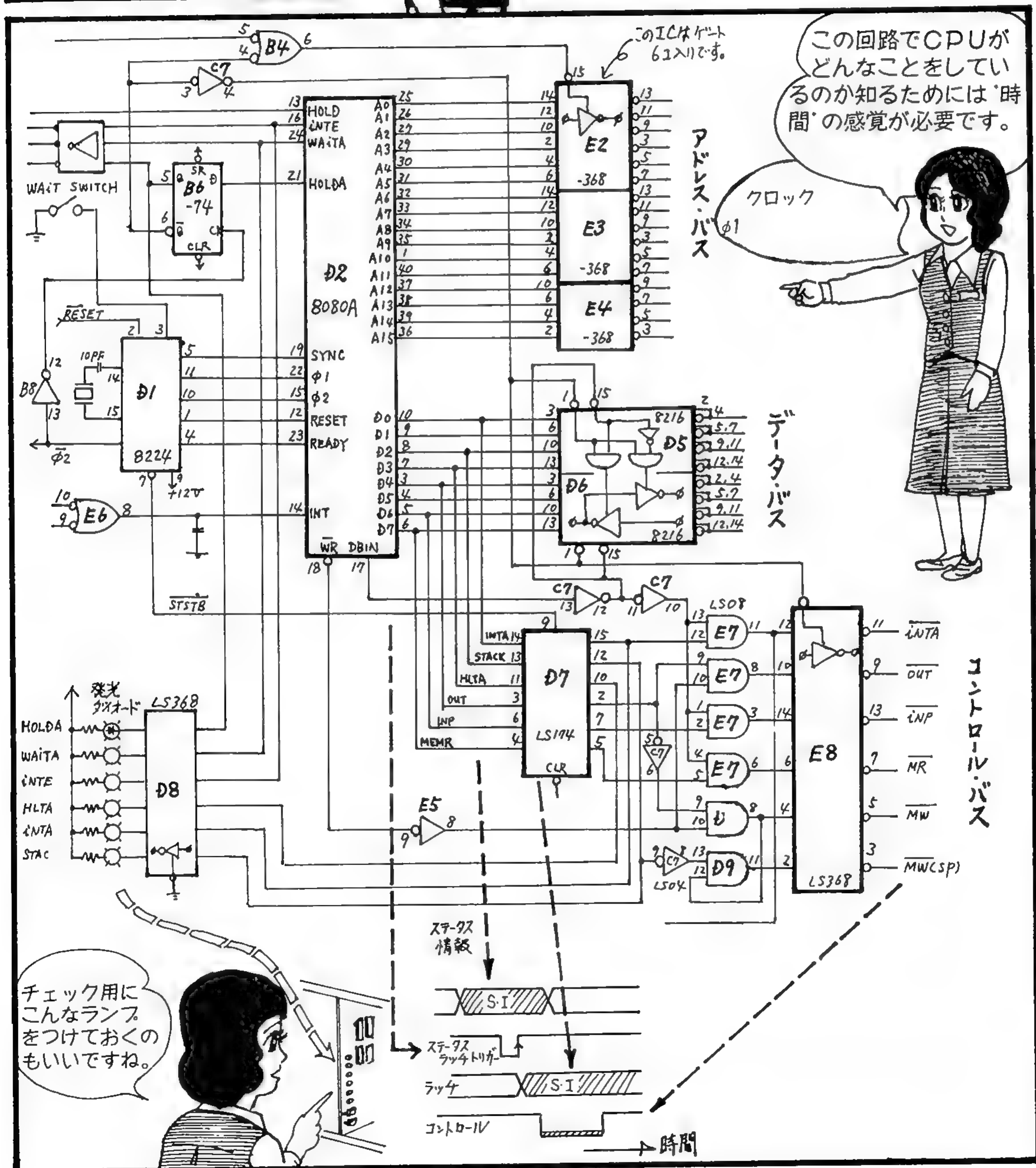
私たちが何気なく使うプログラムに対してCPUの中では驚異的な作業が行なわれています。

オペコード  
フェッチ  
サイクル

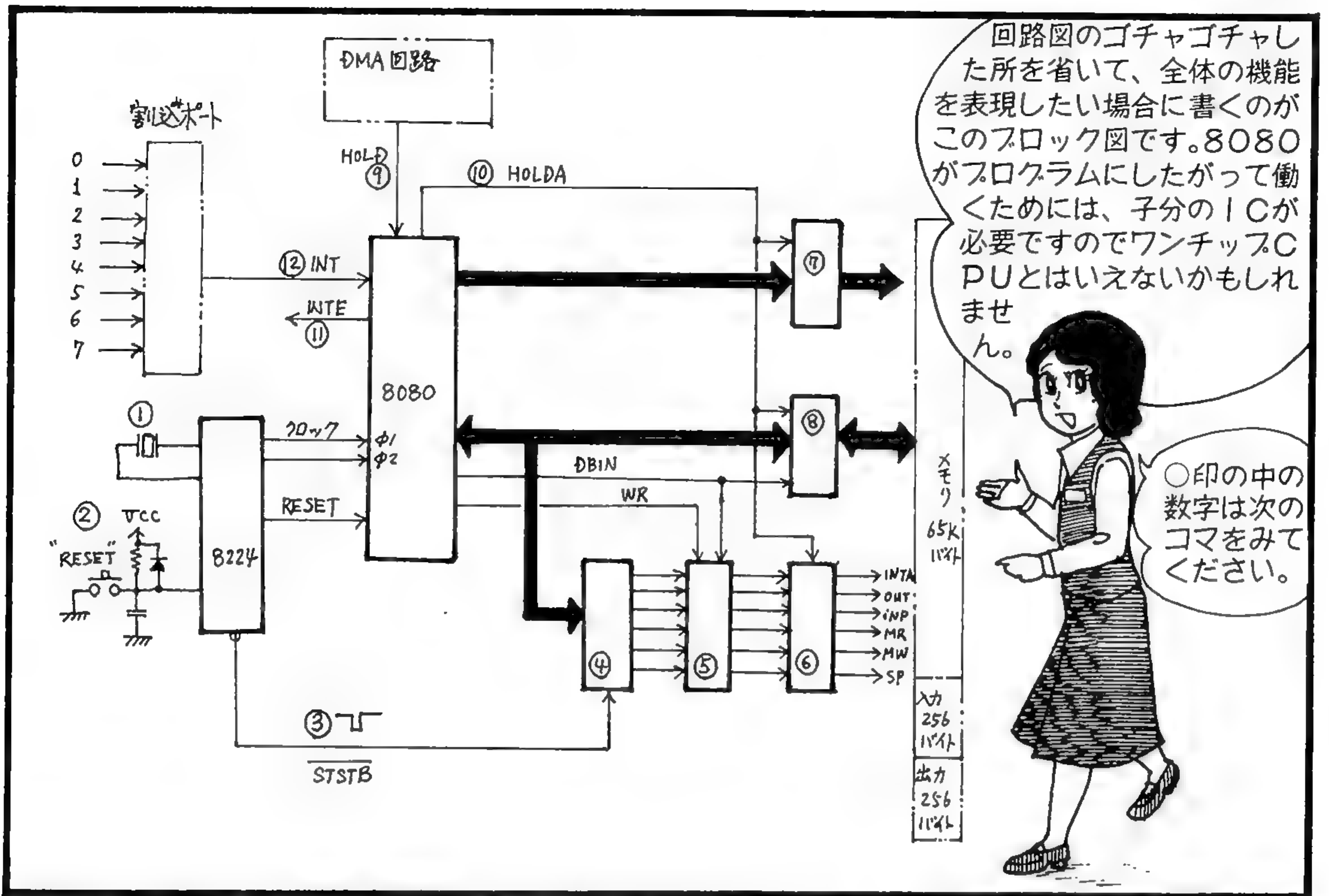


7E  
(MOV A, M)

01111110

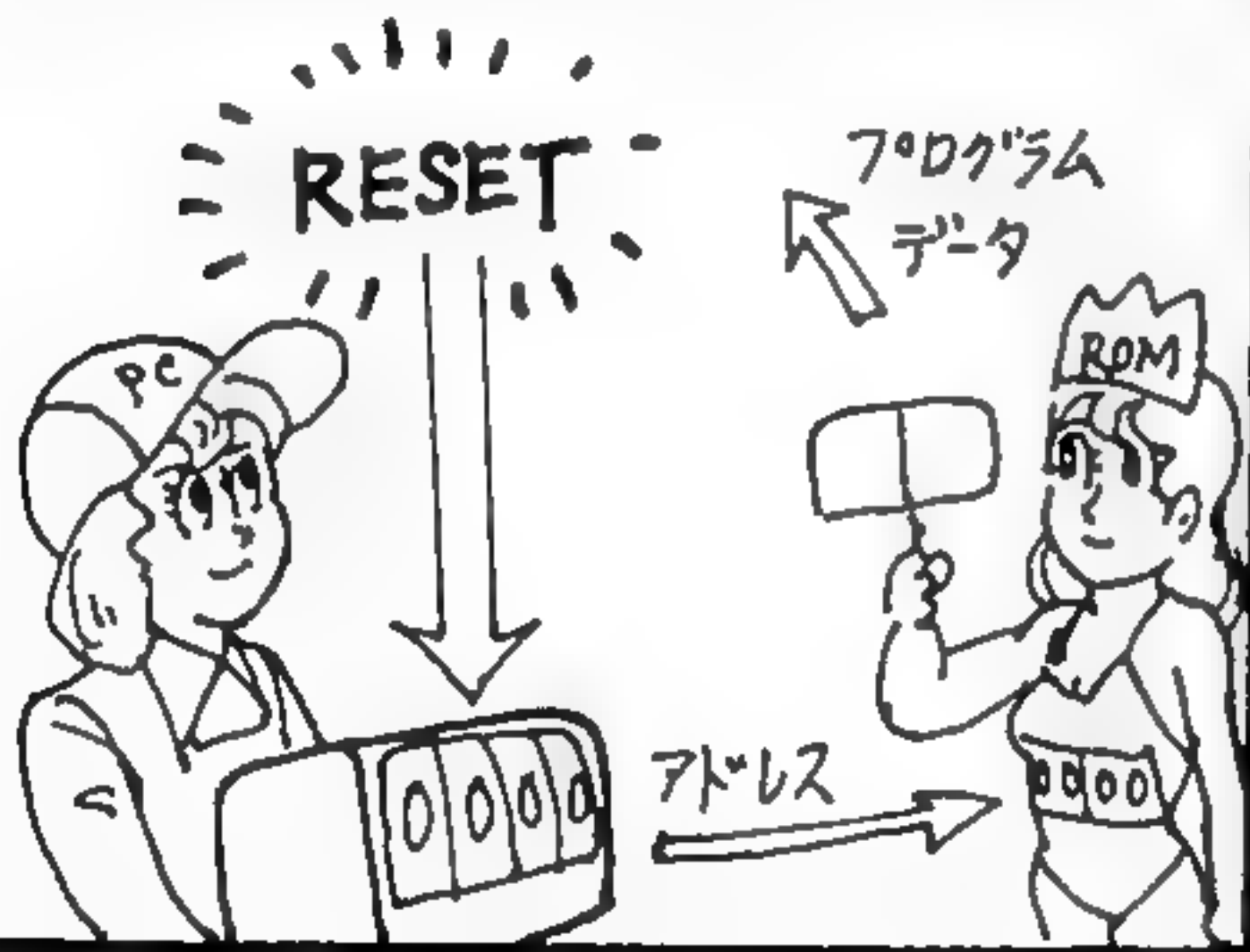






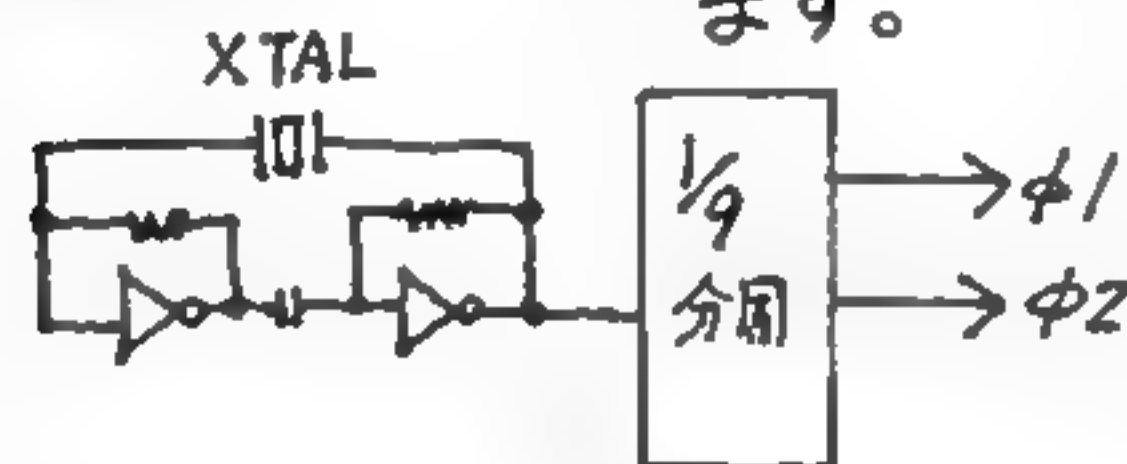
## ②RESET

この信号は、最も強力な割込みです。RESETが入るとプログラムカウンタは0000にクリアされ、プログラムはゼロ番地のオペレーションコードからスタートすることになります。

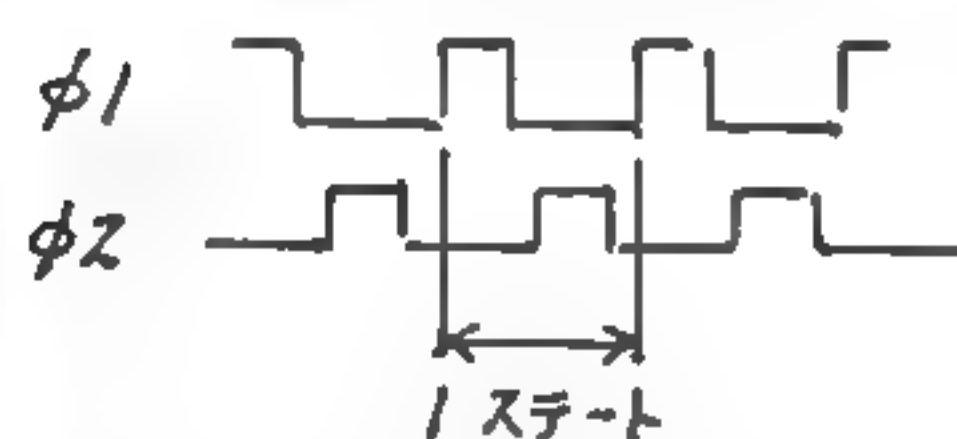


## ①クロック・ジェネレータ

これは水晶発振器で8224と組み合わせてクロックをつくれます。



このクロックの1周期を1スタートといい、マイコン動作時間の最小単位となります。



## ① クロック・ジェネレータ

### ② 電源投入リセット

### ③ ステータス・ストロブ

### ④ ステータス・ラッチ

### ⑤ コントロールタイミング

### ⑥ コントロール(マシンサイクル)

### ⑦ アドレス・バッファゲート(HiZ付)

### ⑧ データ・バッファゲート(HiZ付)

### ⑨ HOLD(HiZ)要求

### ⑩ HOLDA受付

### ⑪ INTE(割込み受付可)

### ⑫ INT(割込み要求)

これだけわかればちょっとしたものです。

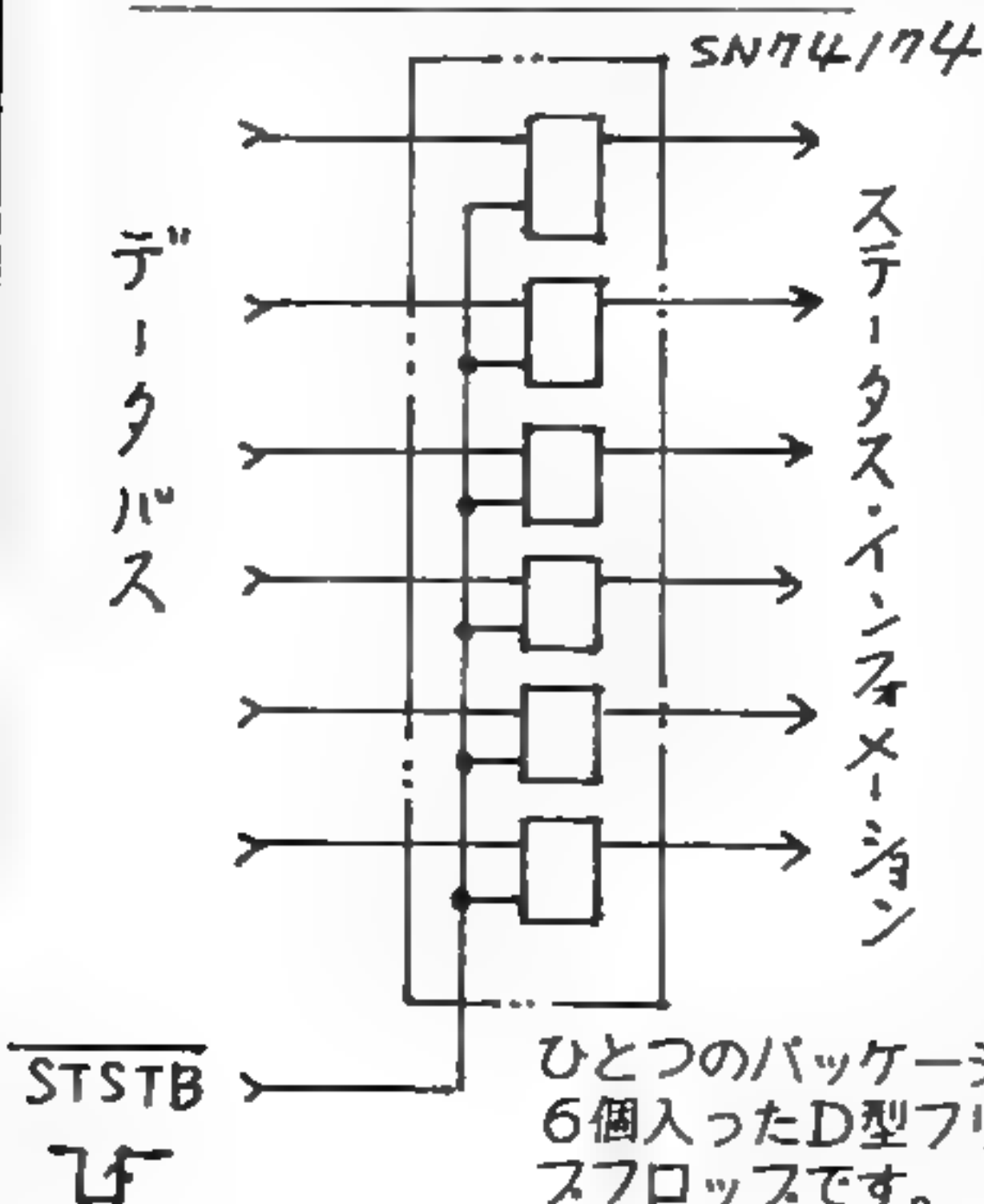
## ⑤コントロールタイミング

ステータス・ラッチはのっぺりしたものなのでROM、RAM、I/Oのコントロールには使えません。CPUから出力されるDBIN、WRと論理回路を組んでコントロール信号をつくれます。

DBIN .... データバスがCPU読み込み方向であることを知らせます。

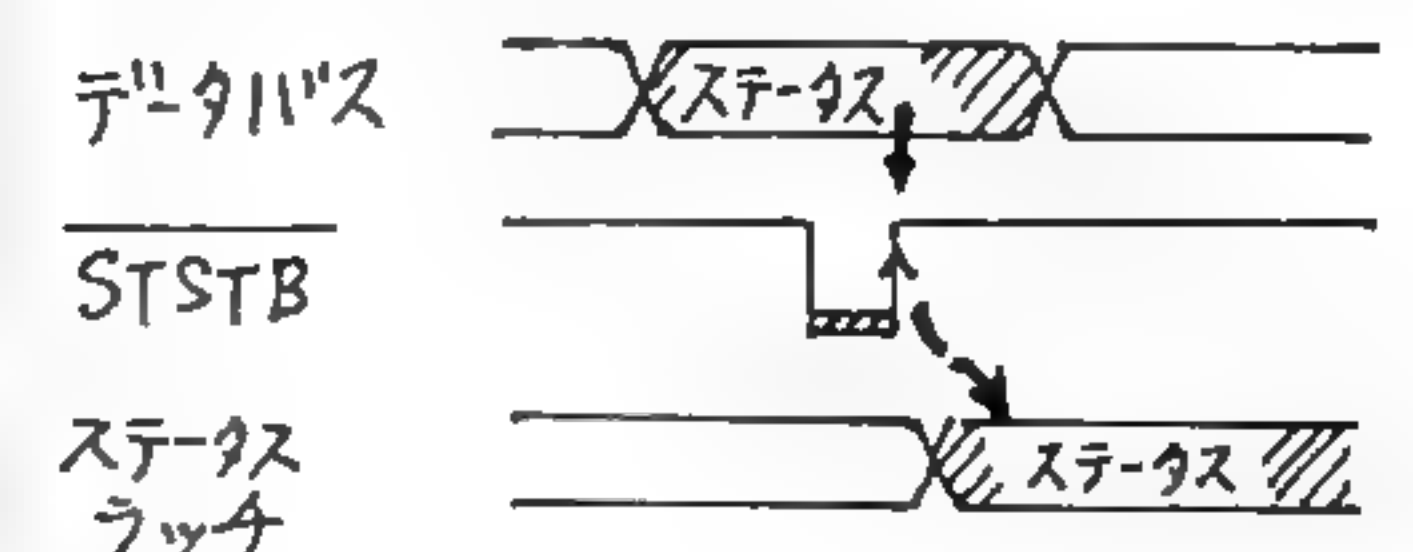
WR .... データバスがメモリにデータを書き込むことを知らせる。

## ④ステータス・ラッチ



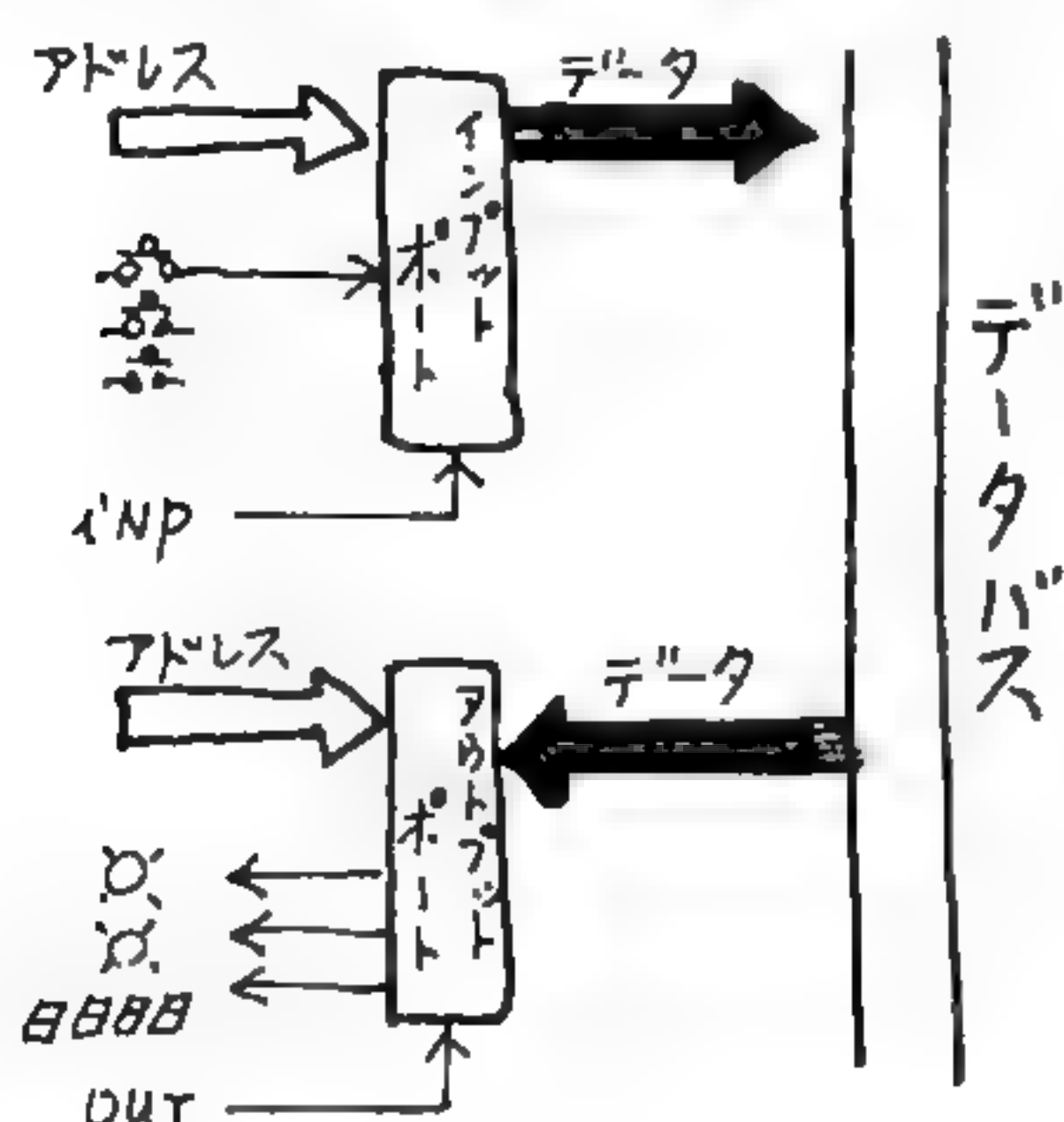
## ③ステータス・ストロブ

データバスにはマシンサイクルの始めにそのサイクルでは何をするのかを知らせるステータス信号というものが出ている。このステータスを外部回路でラッチさせるタイミングをとるのがこのSTSTBです。

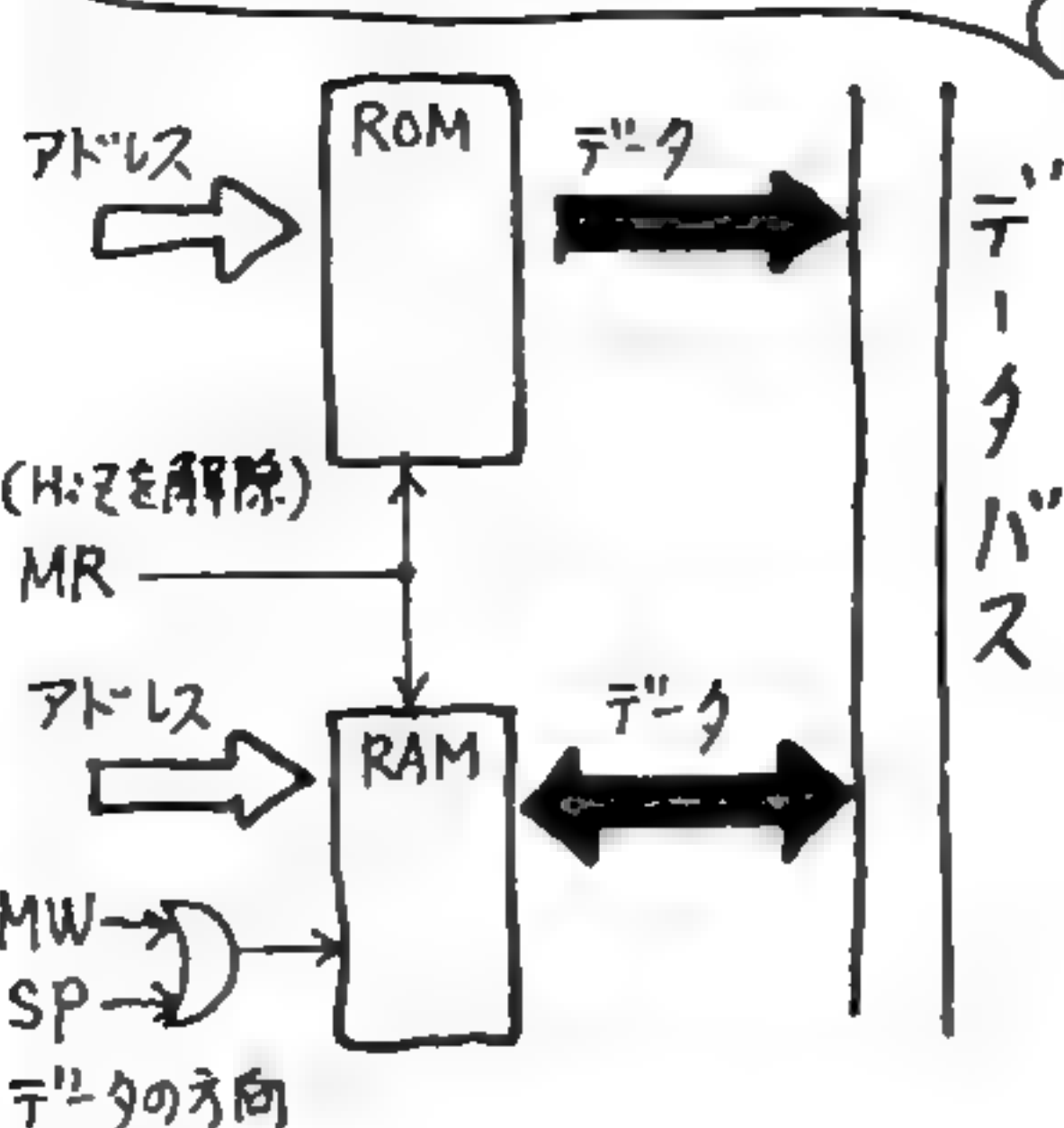




INPUT、OUTPUTの場合は  
inp、outを使います。

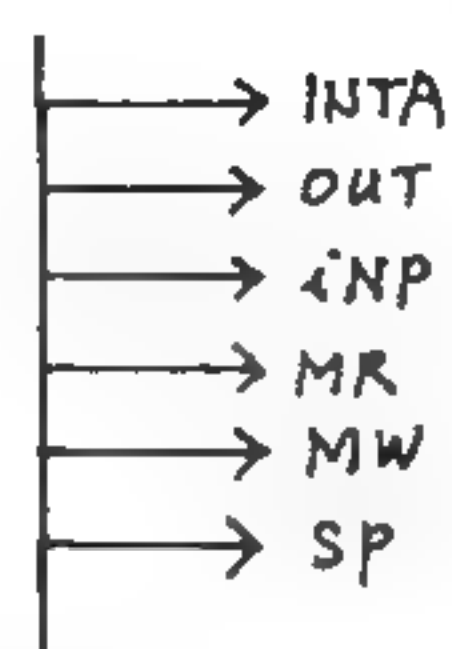


メモリ・デバイスにはMR、MW  
でコントロールします。



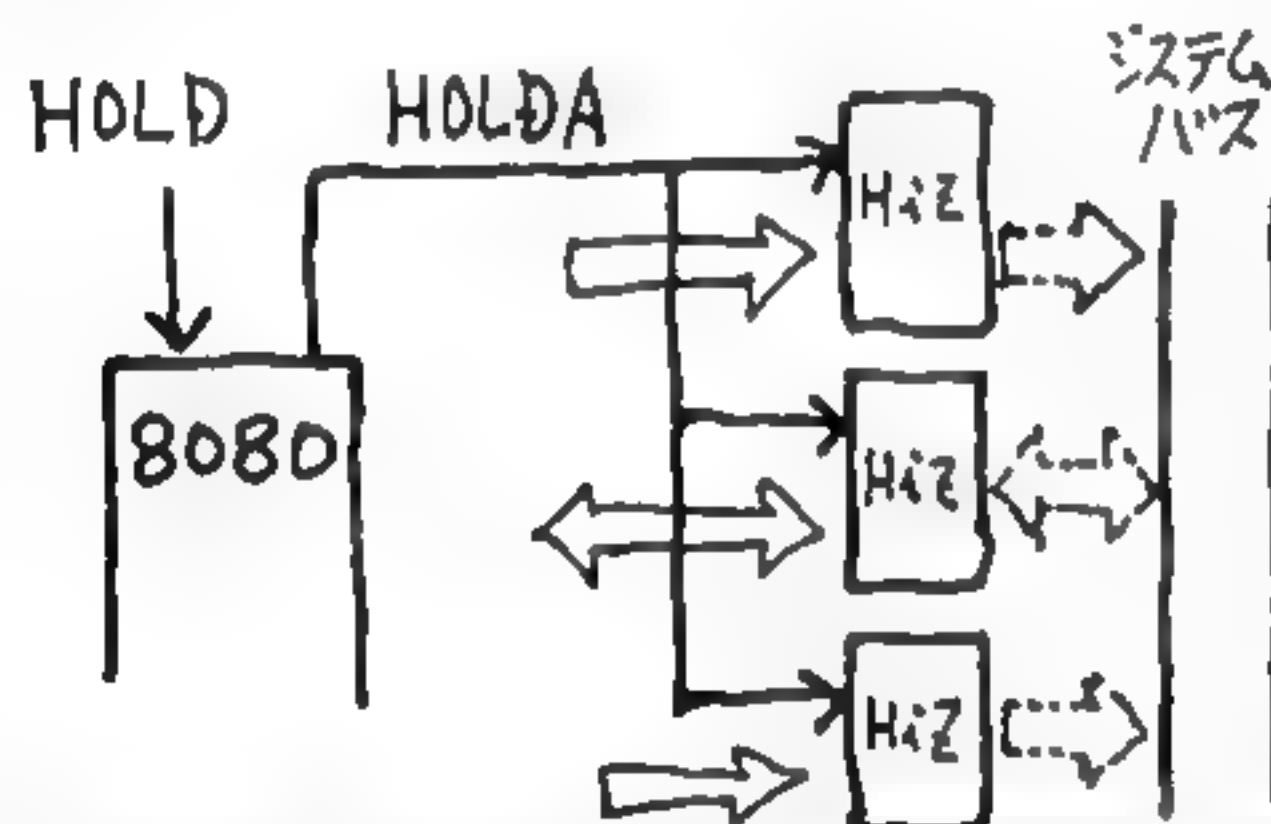
## ⑥コントロール

この回路では6本のコントロ  
ール線がありますが、各々必  
要なデバイスに接続されます。



## ⑩HOLDA

ホールド・アクノリッジホ  
ールド要求を受容した確認信号で、  
CPUはプログラムの実行を停止  
するとともにこの信号を出し続  
けます。回路図ではこれを3つ  
のバスのHiZゲートに入ってい  
ます。

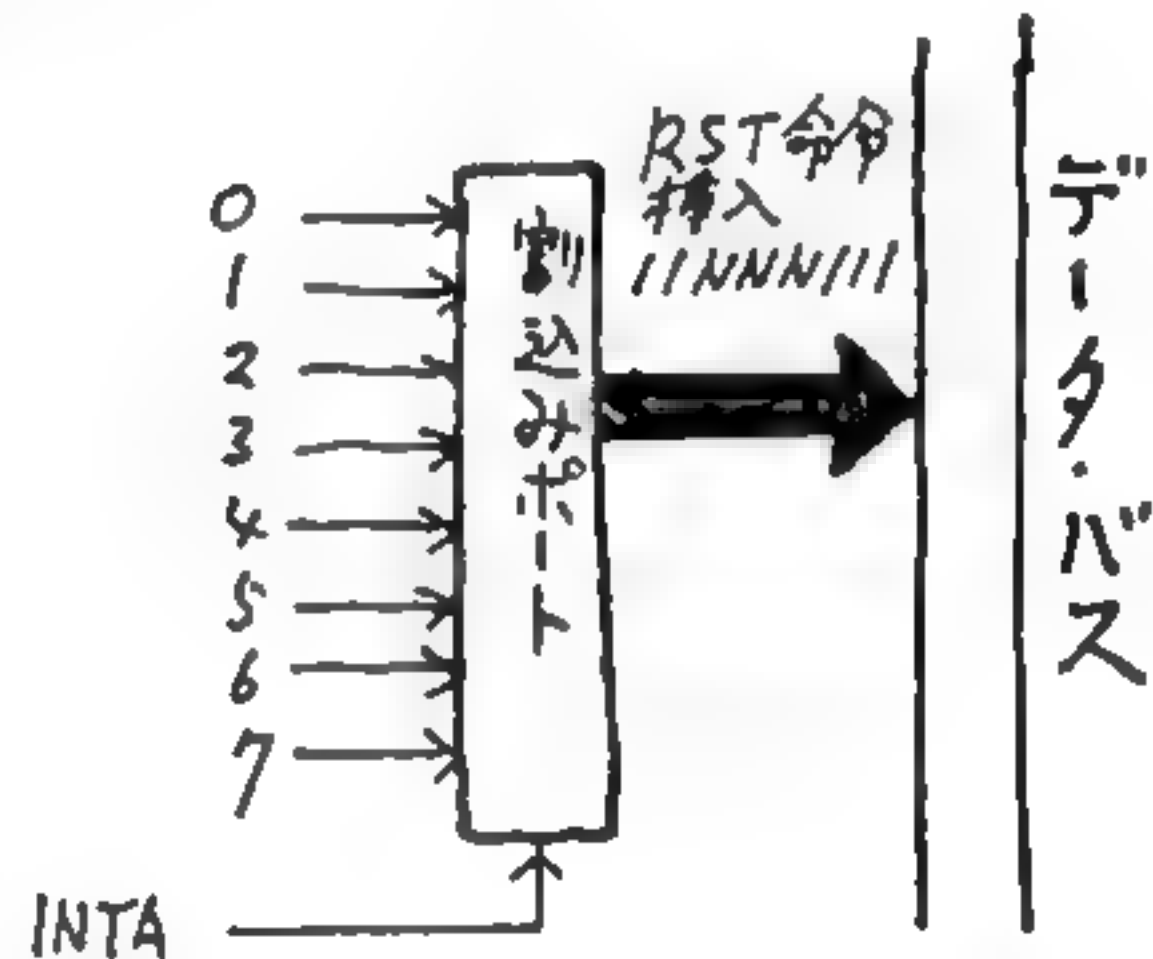


## ⑧HOLD ホールド要求

この信号は、アドレス・データ・  
コントロールの3つのバスをC  
PUの占有から切り離してほし  
いという外部ハードからの要求  
です。



INTAはインタラプト・アクノリ  
ッジの略で割り込み要求が受け付  
けられたことを示し、このマシ  
ンサイクルでデータバスの内容  
をRST 命令として読み込まれ  
ます。



INPUT、OUTPUT、INTに  
ついては、メモリ65Kバイトの  
ほかに、ポートという256、256、  
1バイトの専用メモリエリアが  
あるのです。

メモリ	インポート	アウトポート	割り込み
65K	256	256	1



## ⑫INT

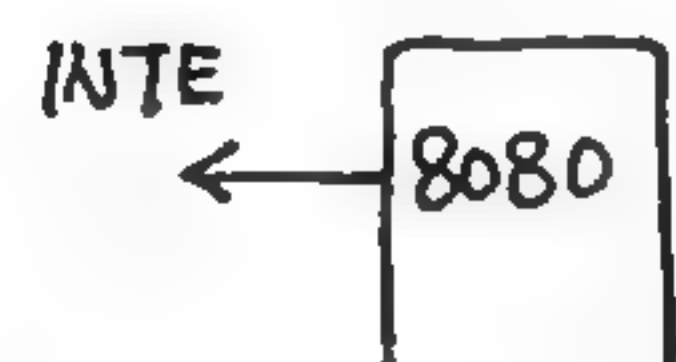
インタラプト要求  
外部から回路の割り込み要求端子  
です。CPUがINTE状態にある  
と要求を受け付け、次のマシ  
ンサイクルでステータスにINTA  
を出力します。



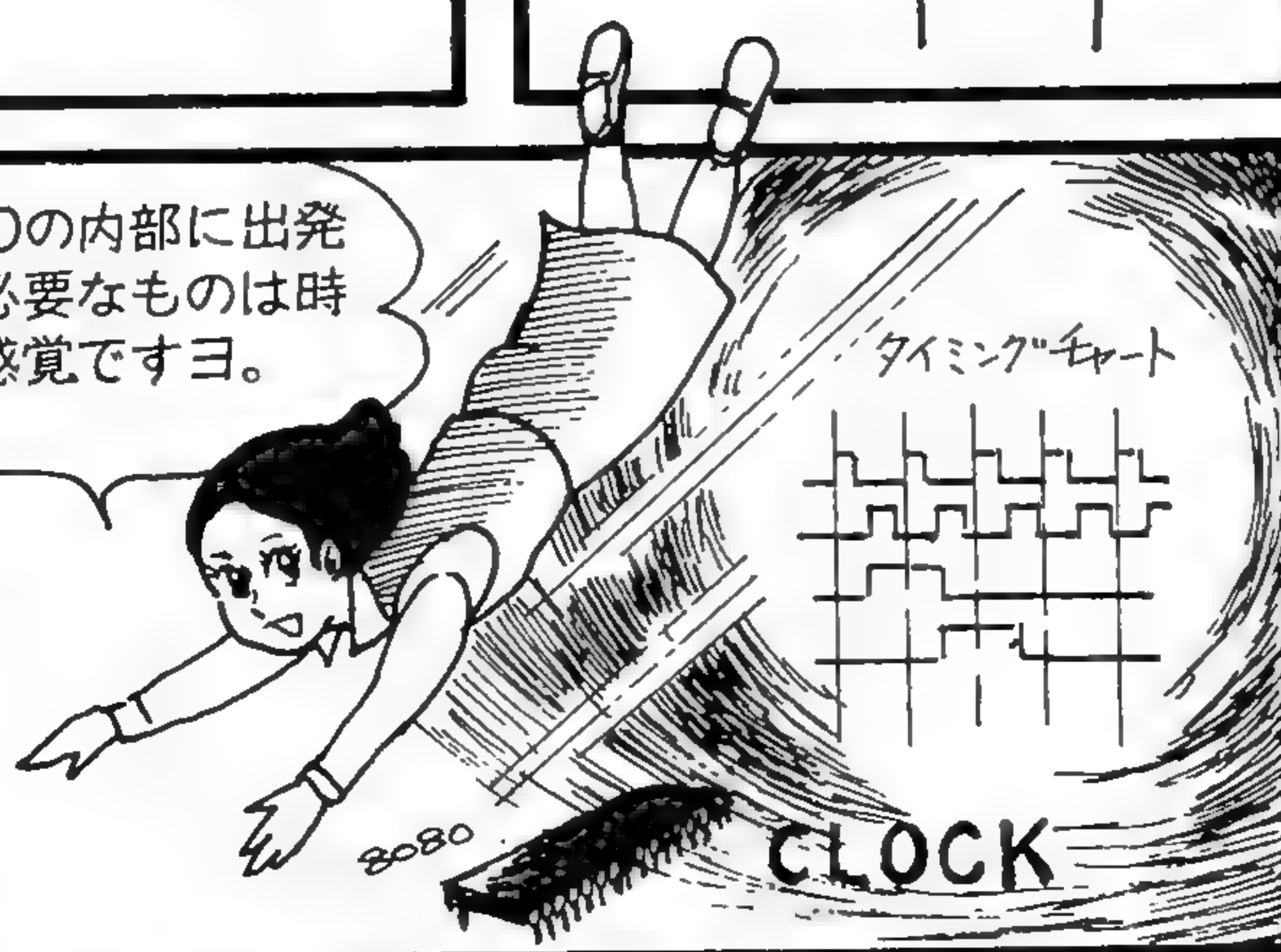
## ⑪INTE

インタラプト・イネーブル

この信号は、プログラムでEI  
命令を実行させる必要がありま  
す。これが出ている時はCPU  
が割り込み受け状態にあるとい  
うことです。



では、8080の内部に出発  
です。最も必要なものは時  
間に対する感覚ですヨ。



この3つのポートについては、  
8080タイミング説明とは別に  
お話することにします。

インポートポート  
アウトポートポート  
インタラプトポート





# ⑨CPUとクロック



MOV A, Mの実行

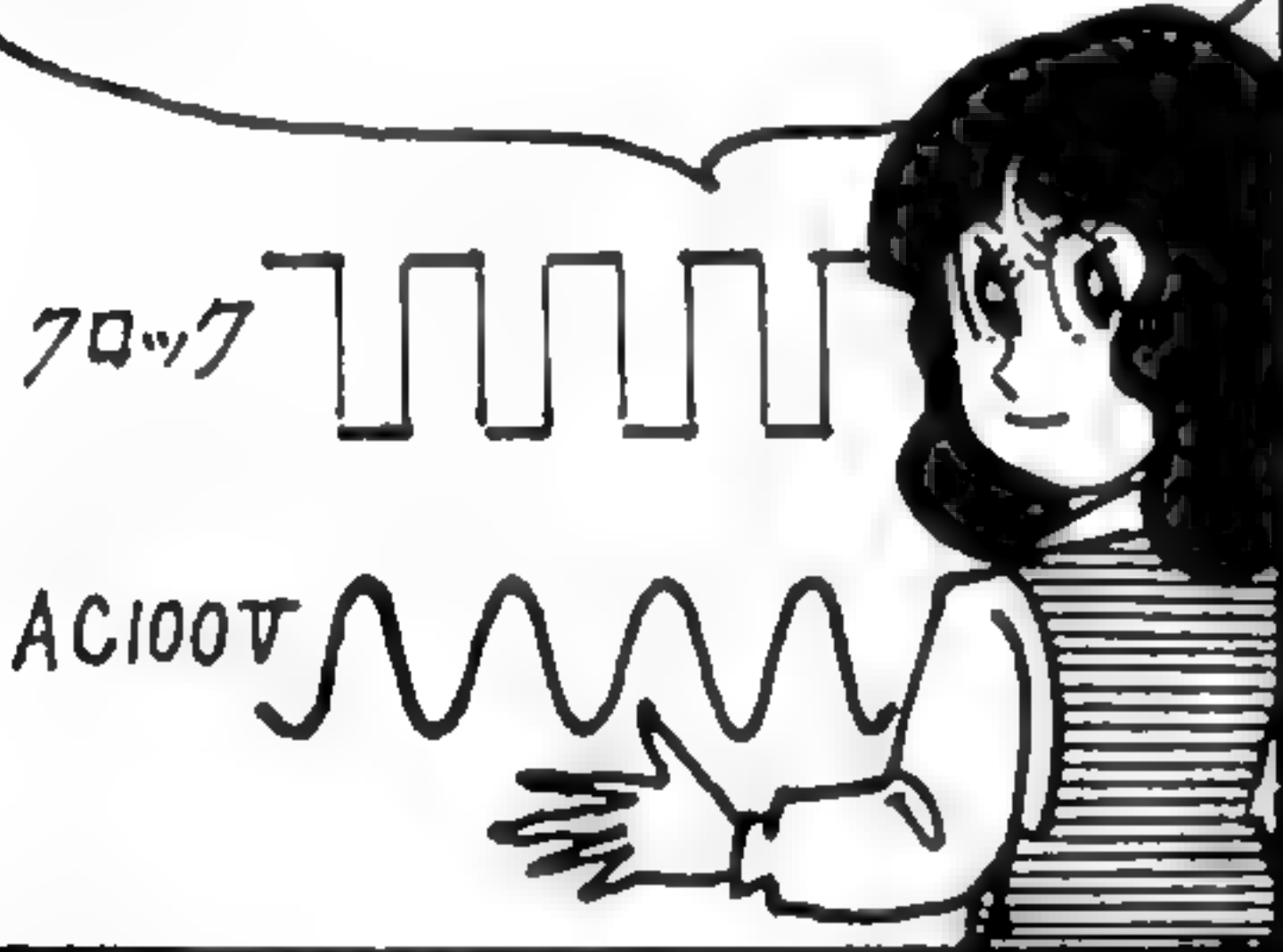
M1...オペコードフェッチ  
サイクル

M2...実行サイクル

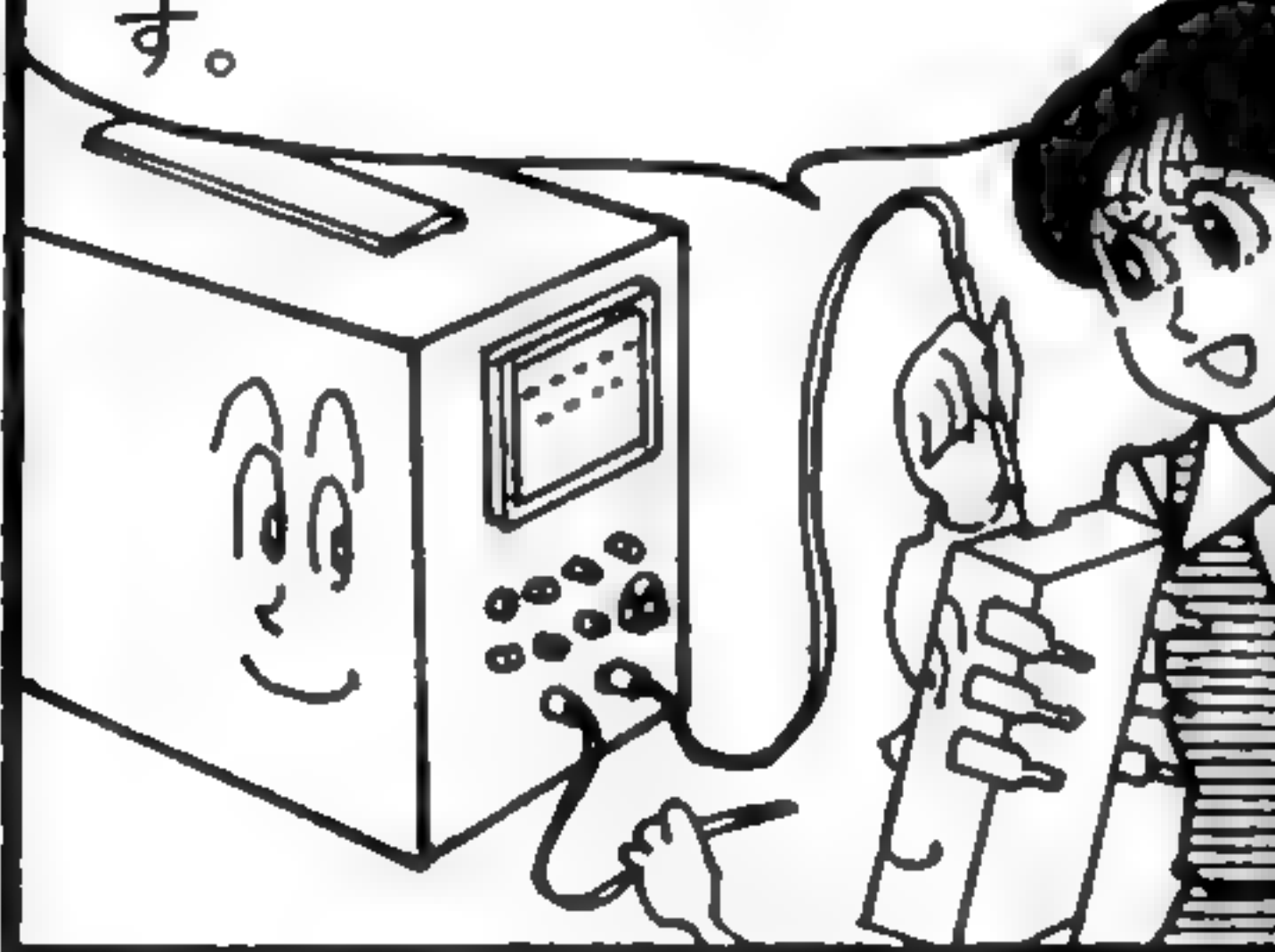
M1			M2			M3	
T1	T2	T3	T1	T2	T3	T1	T2
オペコードの フェッチ (取込み)			(A)←(M)の 実行				

MOV A, Mを  
実行するのに6つ  
のクロックが必要

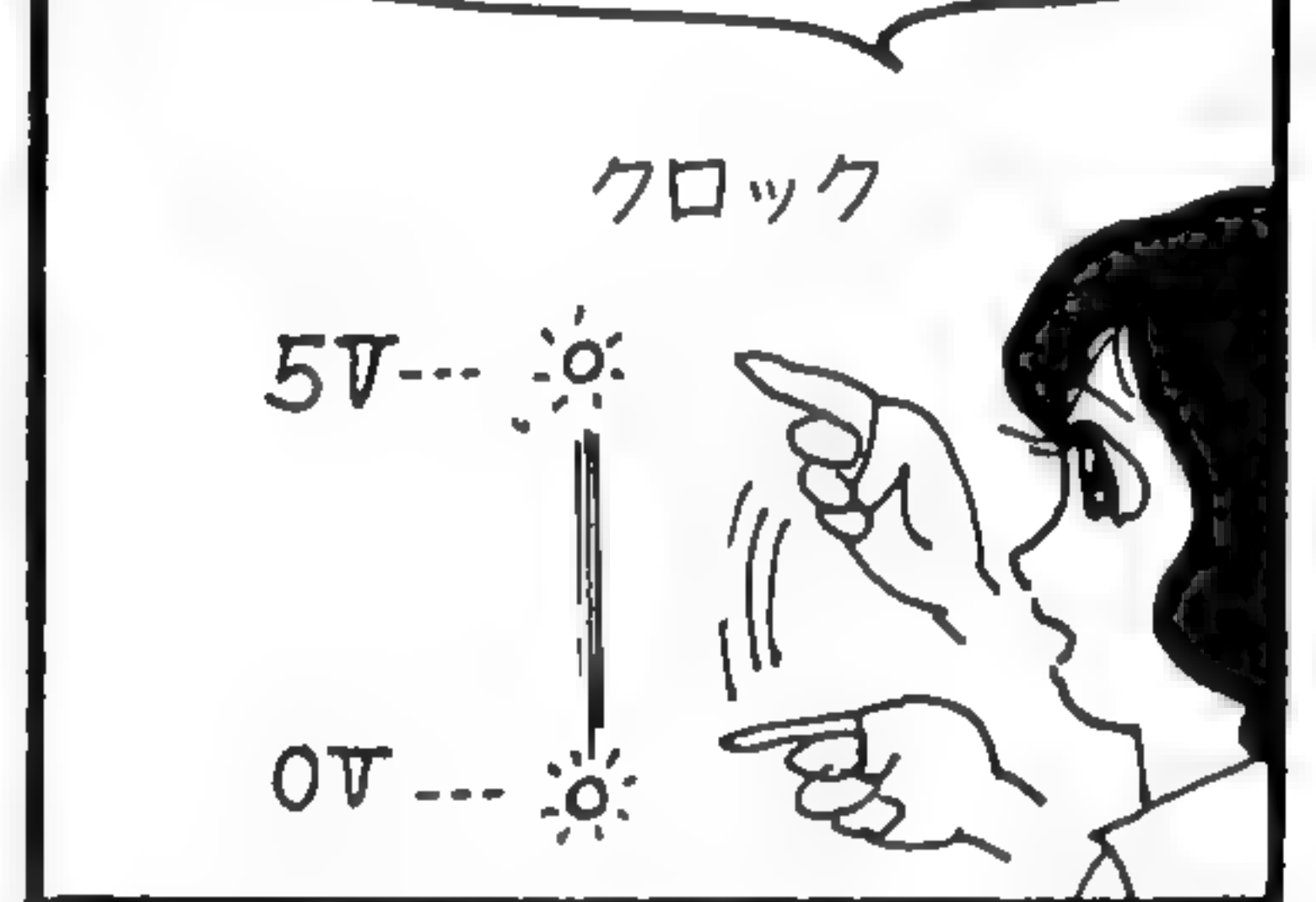
この図形化されたクロックの  
波形を見る時、次のことにま  
どわされないようにしましょう。



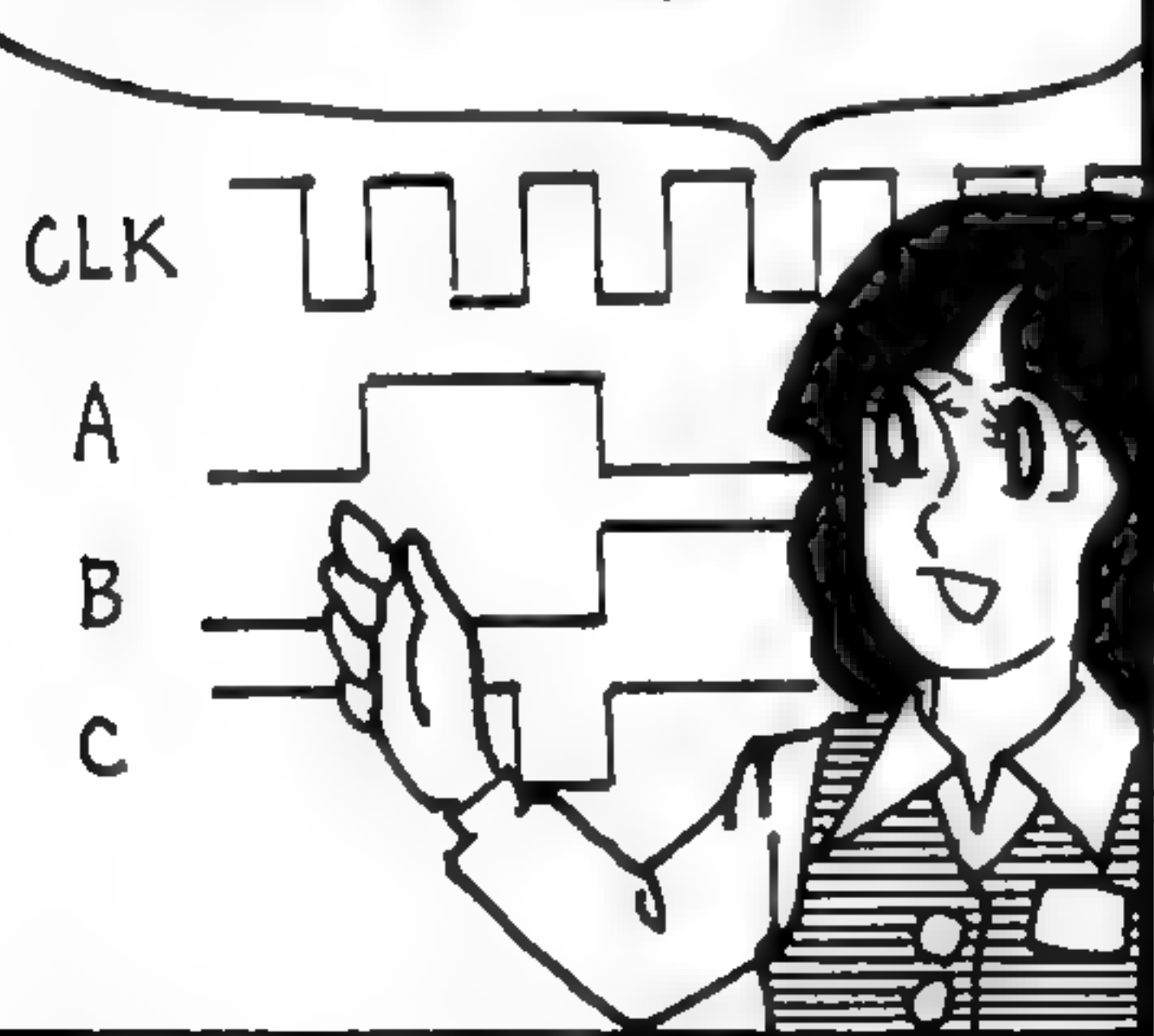
シンクロスコープで見ると、  
回路に のような電圧  
の変化があることがわかりま  
す。



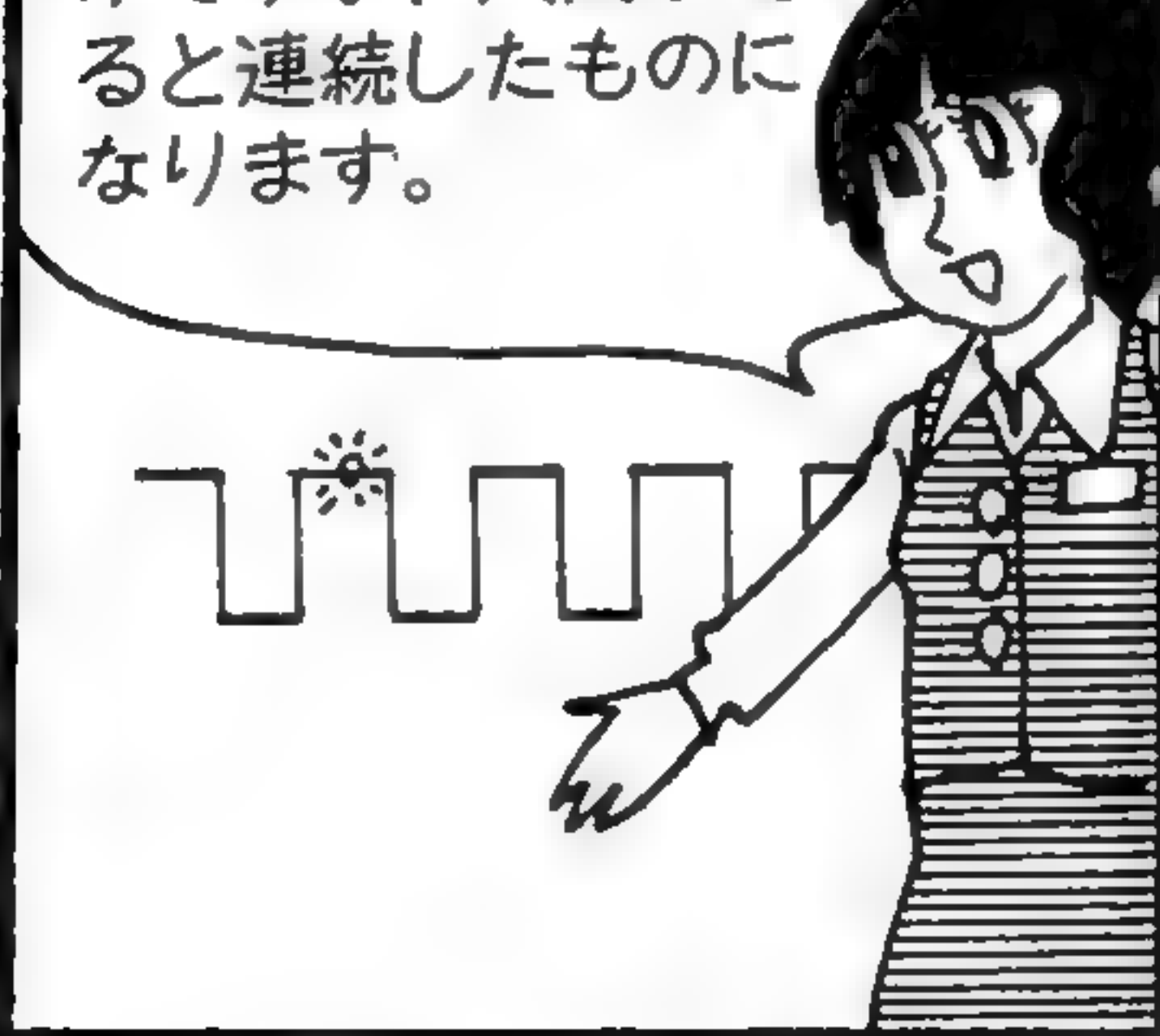
回路的には、電圧が規則  
正しく現われたり消えたり  
します。



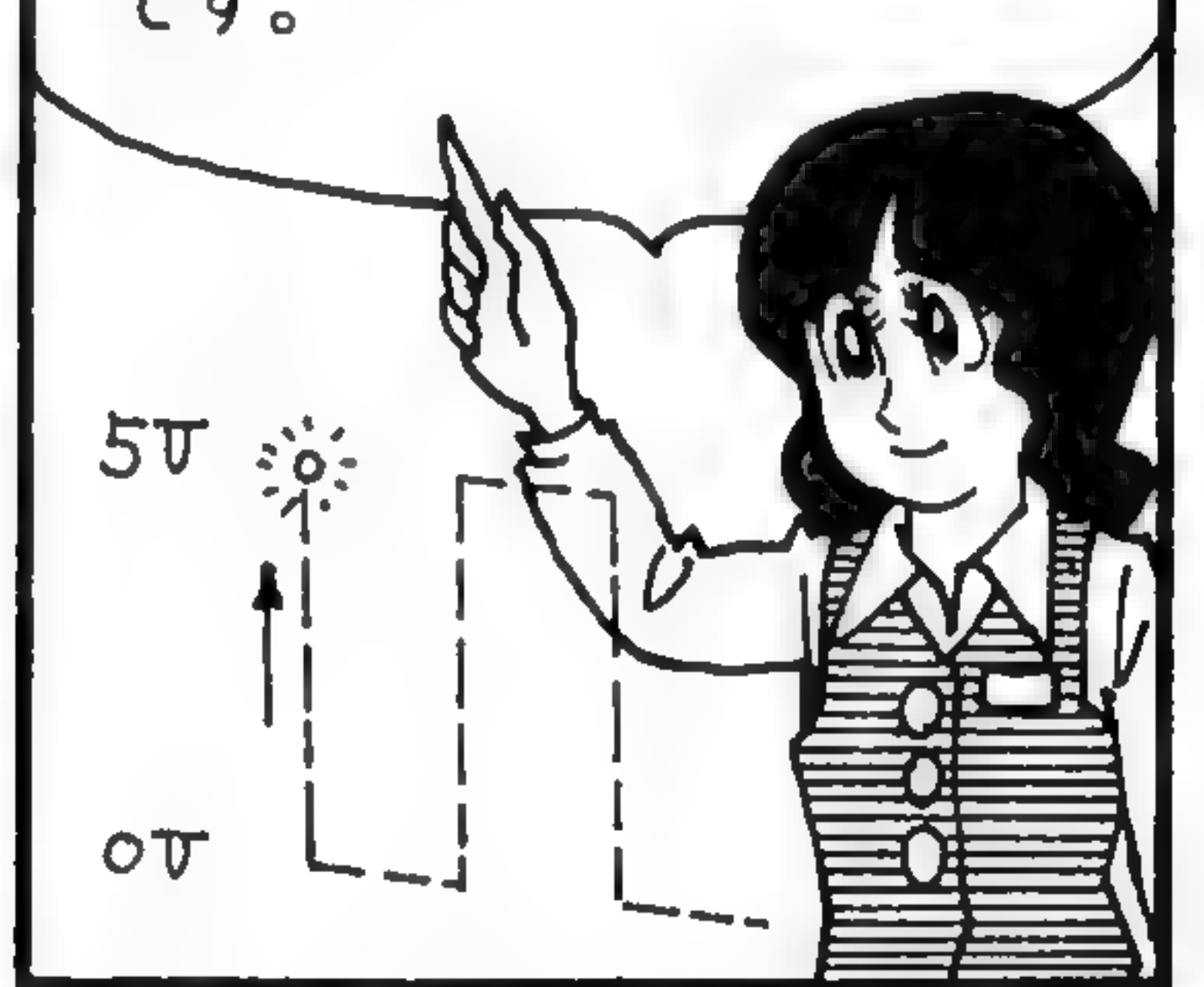
タイミングチャートからは、  
ある時間のできごとを読み取  
るんだと思うと楽ですよ。



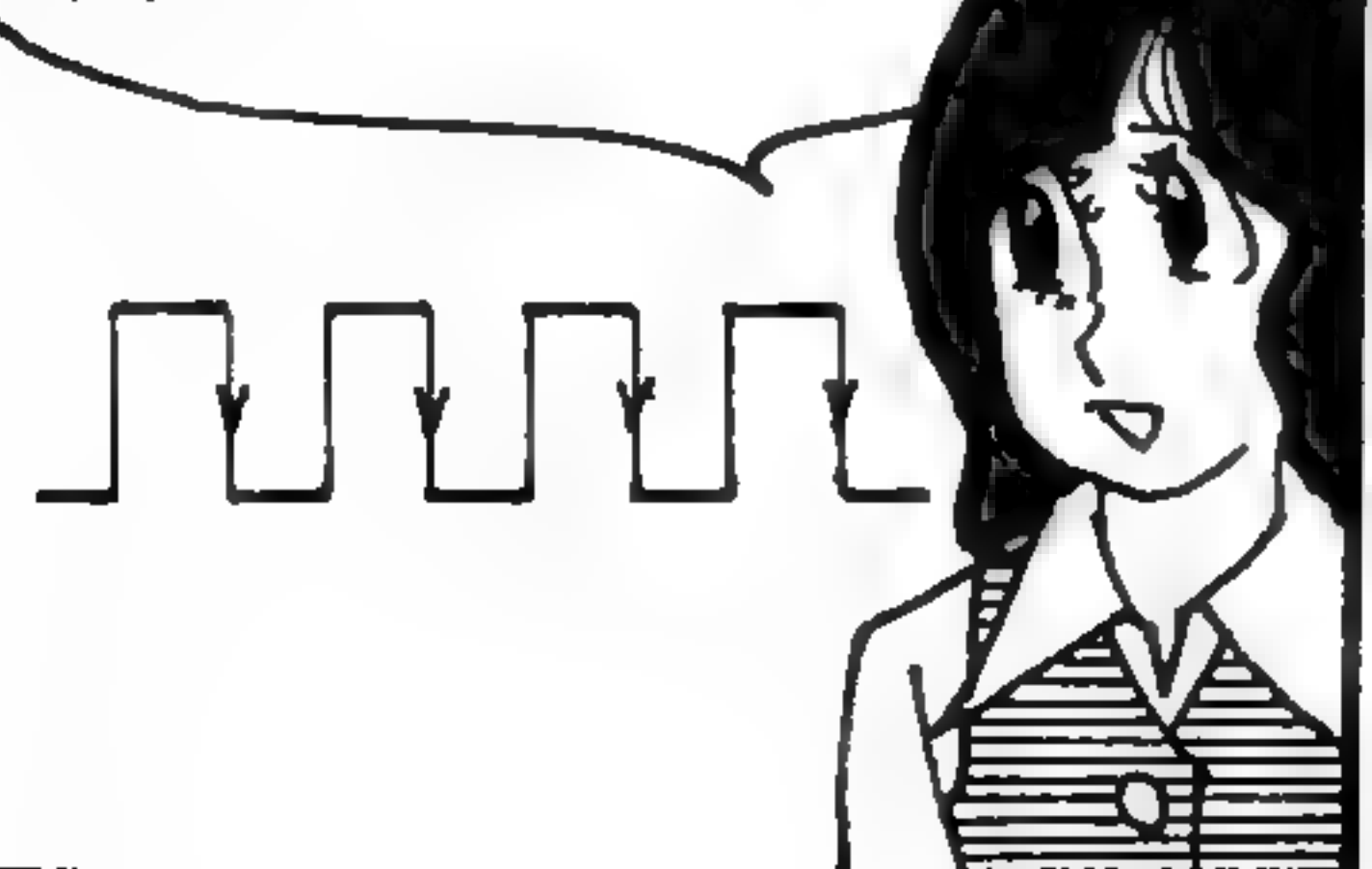
シンクロスコープの場合も、  
ひとつの点(●)が動いているだ  
けですが、人間が見  
ると連続したものに  
なります。



それは、ある時間には一点に  
しか●がないということ  
です。



この端の急激な変化は、微分  
回路でトリガーをつくること  
ができるからです。この図  
はクロックとトリガーを  
併記したもので  
す。



この端が、回路  
的には重要で  
す。

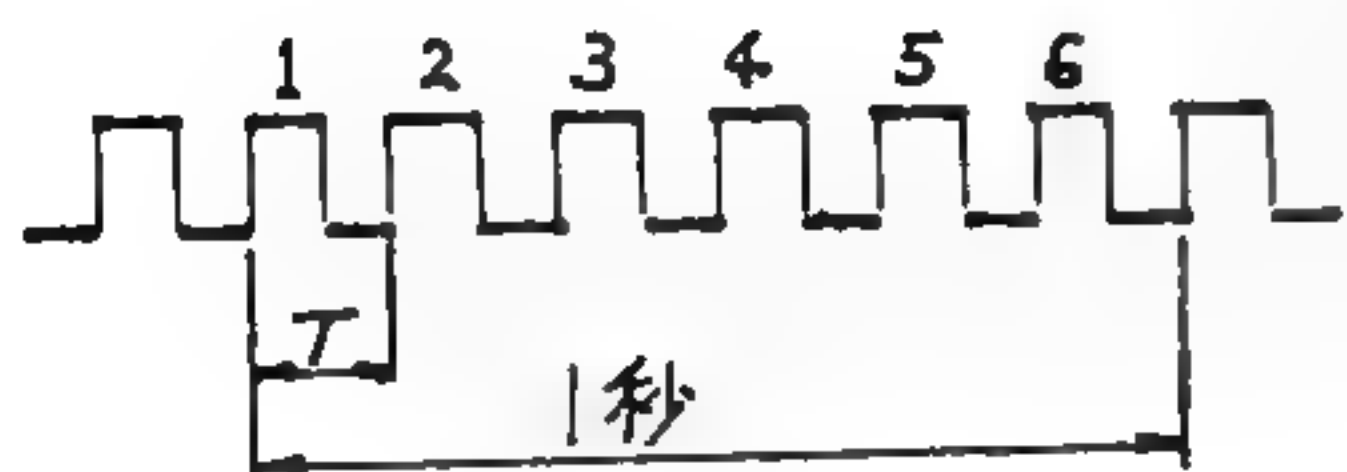


クロックを拡大すると、  
ひとつの山があり、両端  
があります。





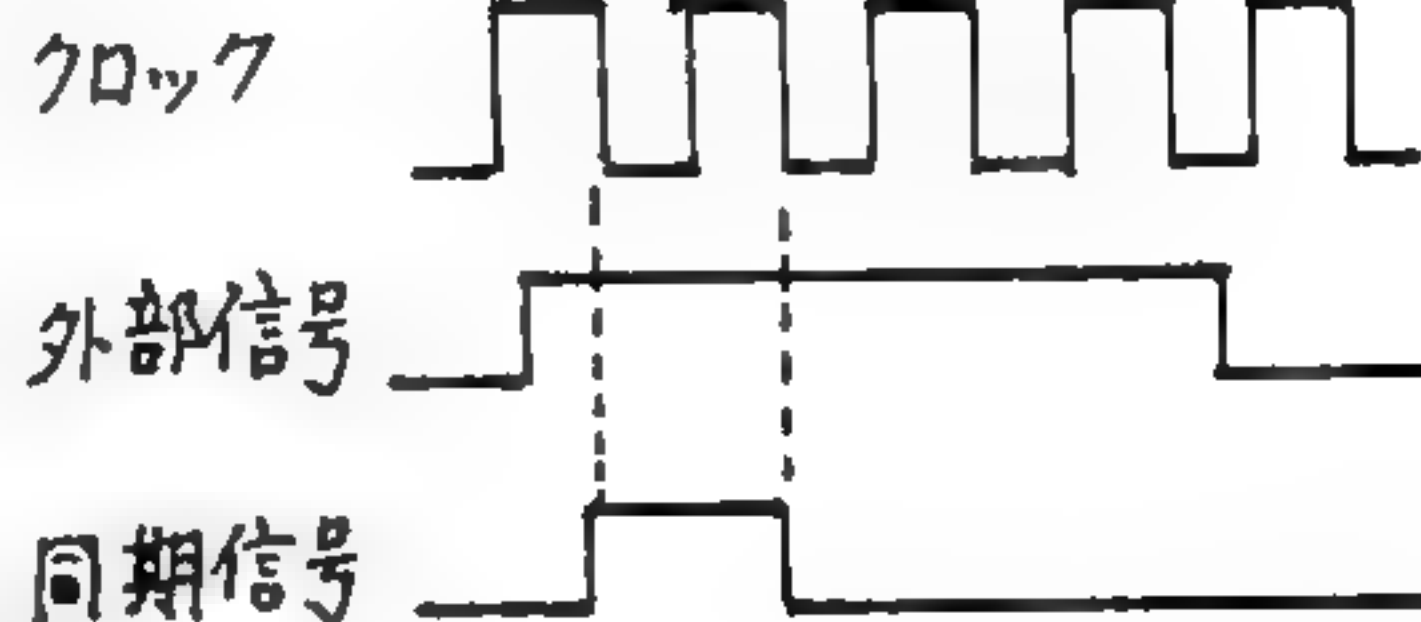
1秒間に何周期あるかをHz(ヘルツ)という単位で示します。図の場合は、6Hzとなります。



クロックのひとつの繰り返し時間を周期Tで表わし、



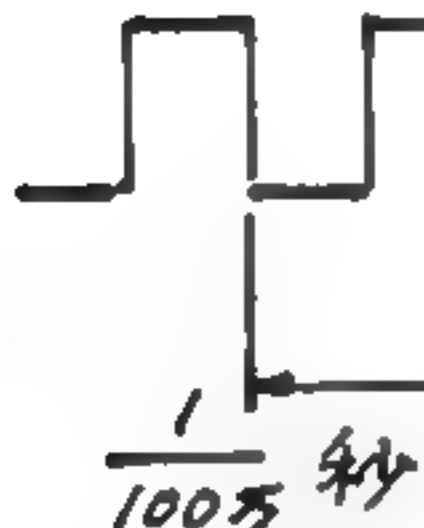
外からきた信号をクロックと同期させ、マイコンが扱いやすい信号に整形することができます。



マイコンでは音を出すことができますが、秘密はこのクロックの速さにあります。



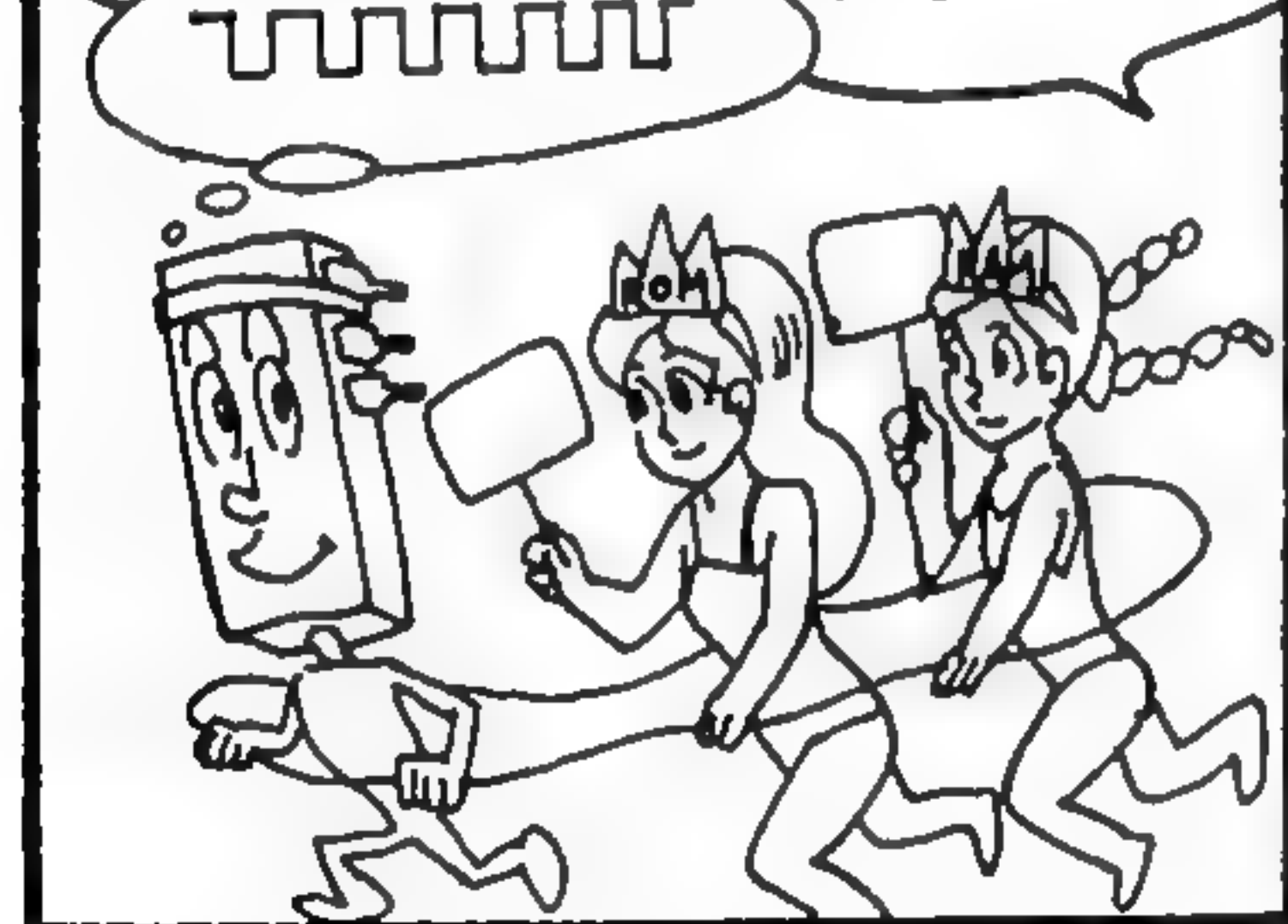
1MHz(メガヘルツ)は、100万Hzのことです。パルスの幅は  $\frac{1}{100万}$  秒です。



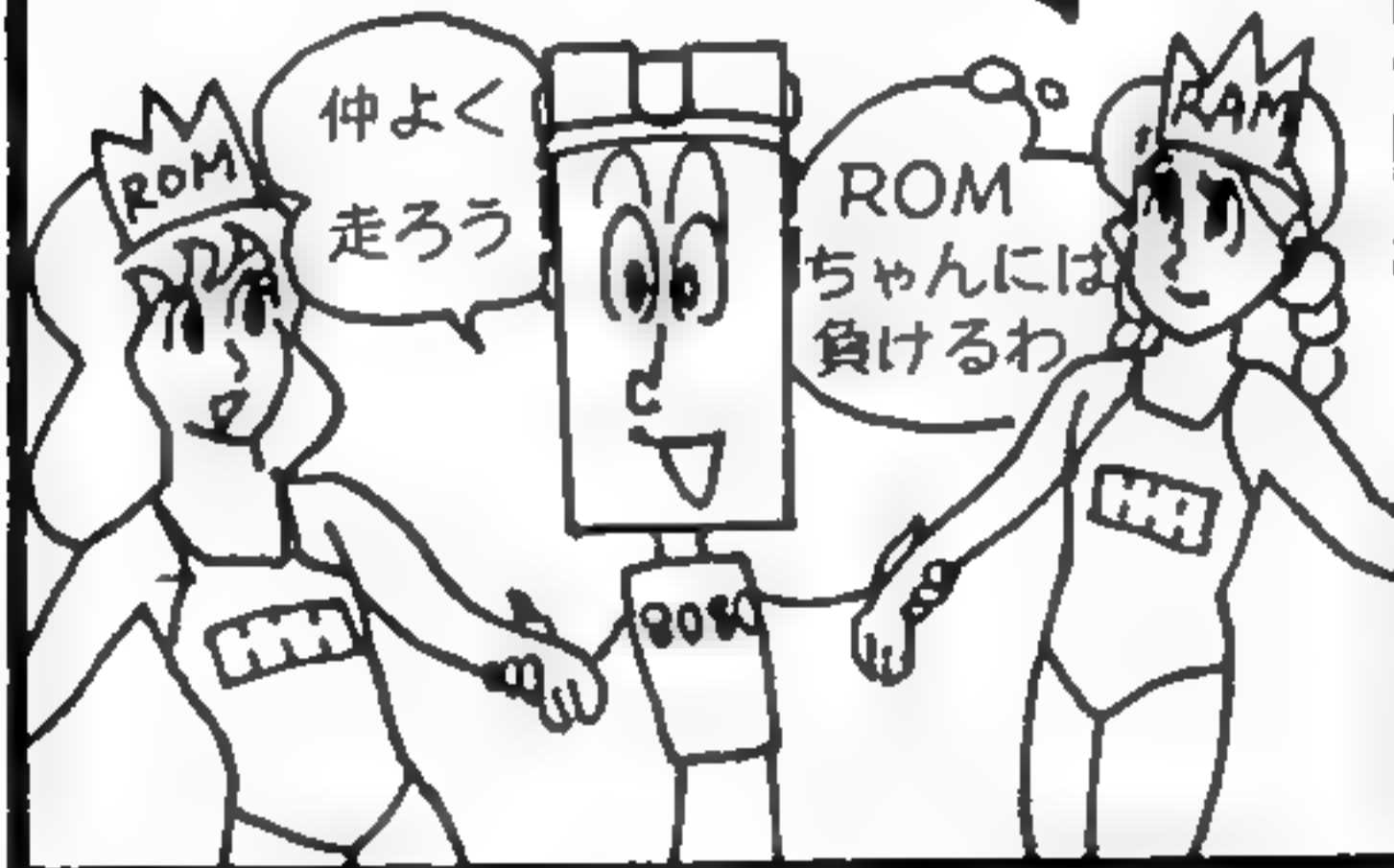
マイコンのクロックは1MHzから4MHzです。



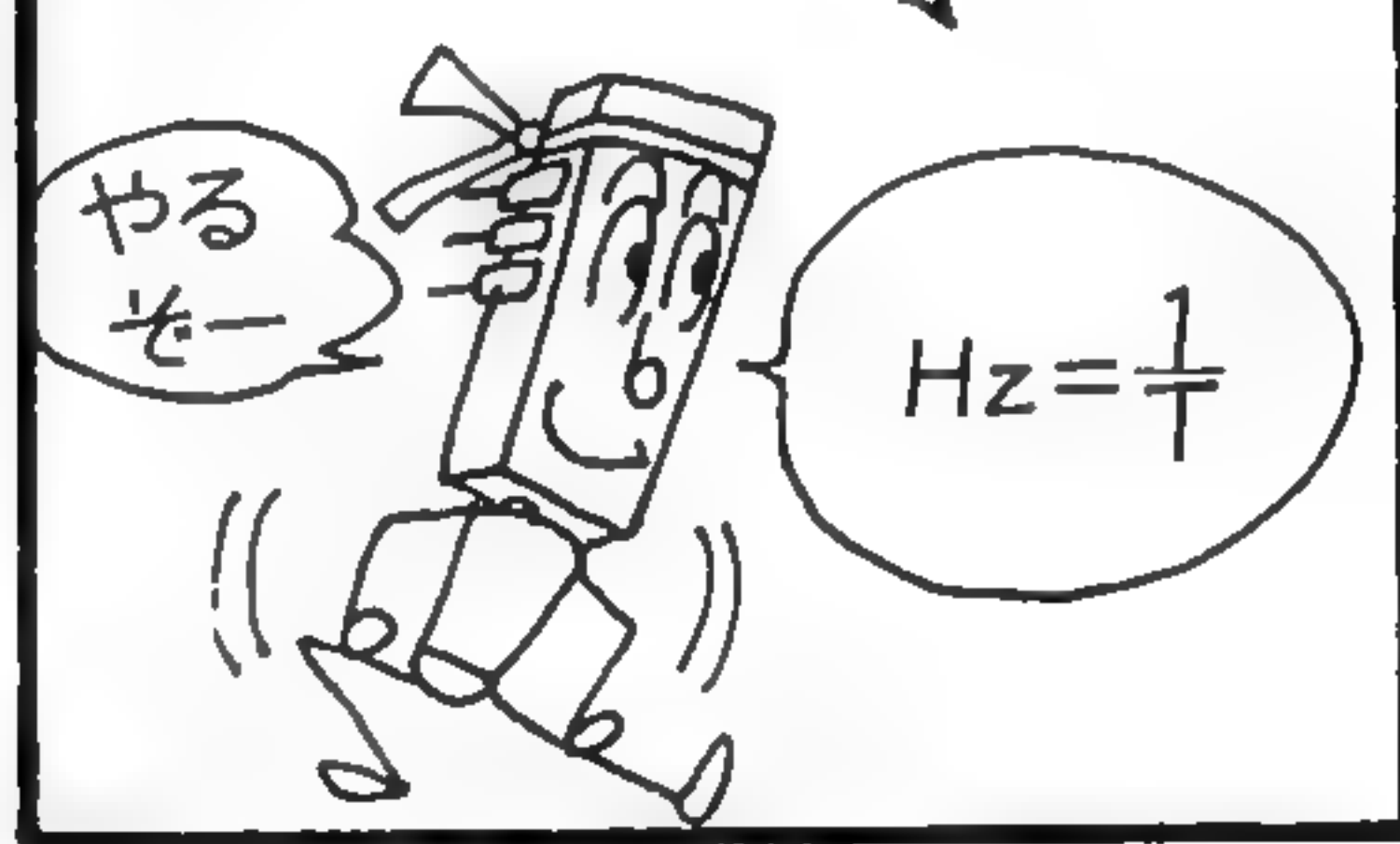
メモリから読み込む時、また書き込む時、内容が違って伝達されたらなんにもなりません。



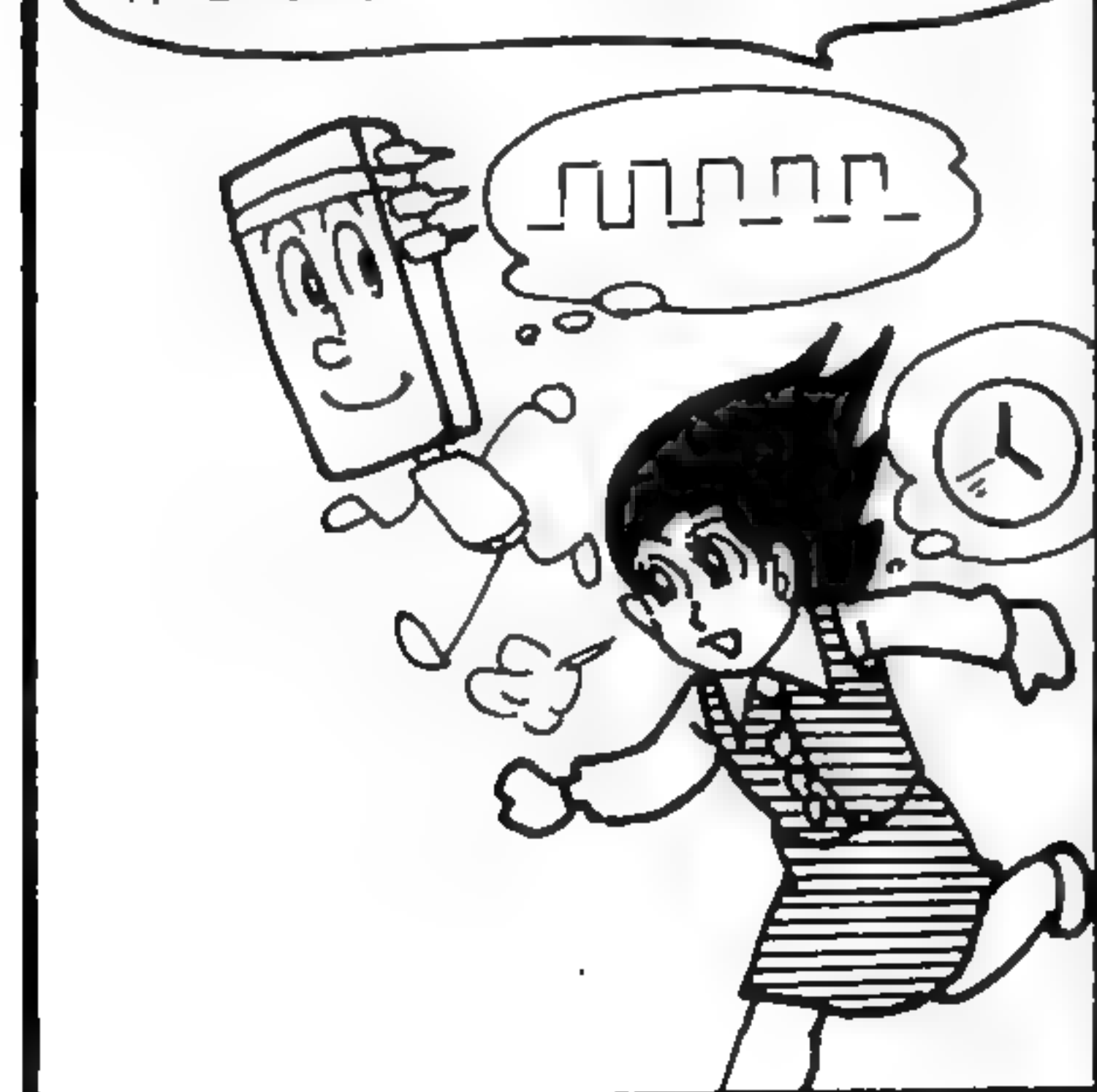
このクロックは勝手に決めて良いわけではなく、CPU、RAM、ROMの処理スピードを考えて決められます。



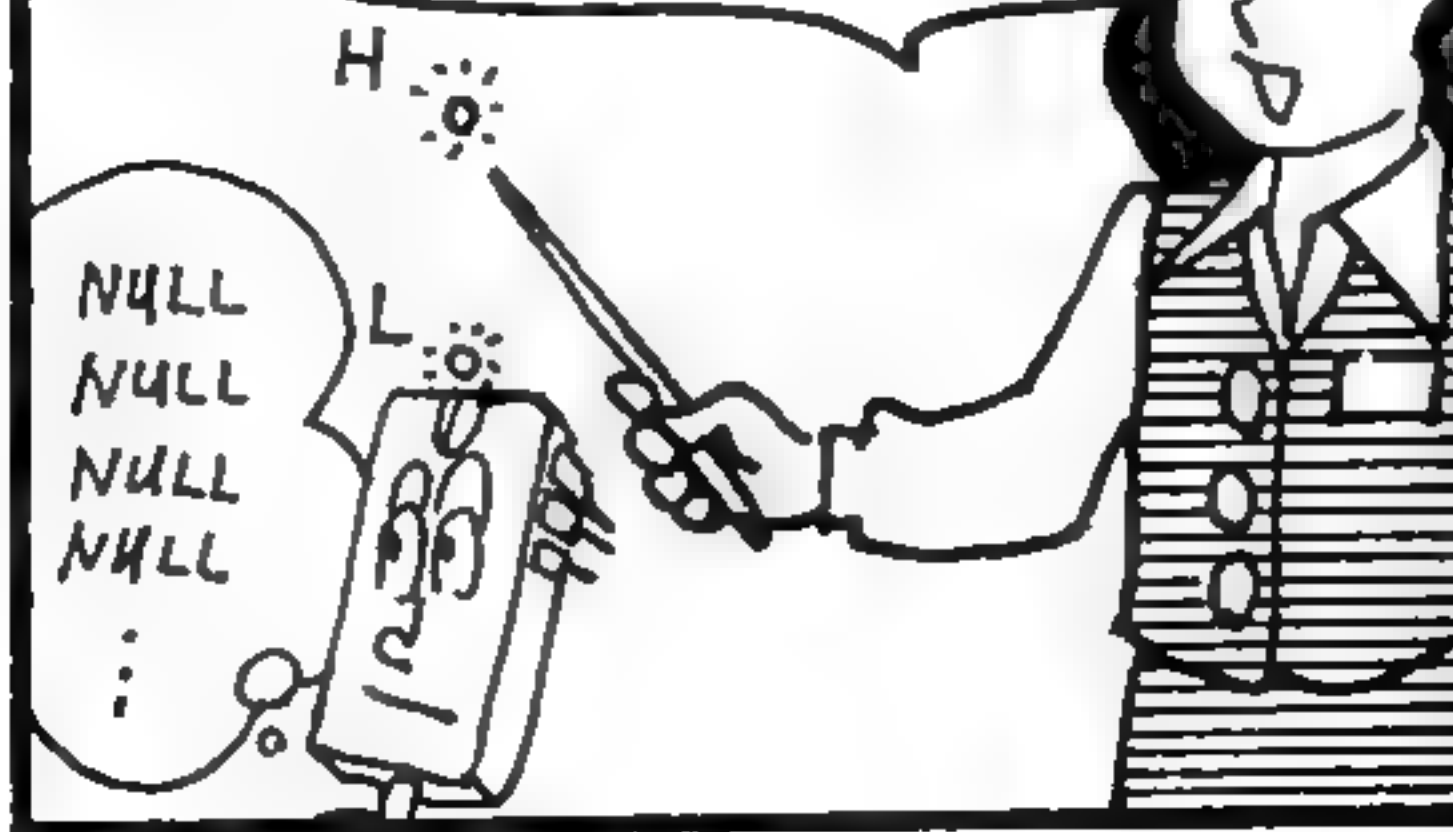
コンピュータの演算速度はクロックが何Hzかによって決まってきます。



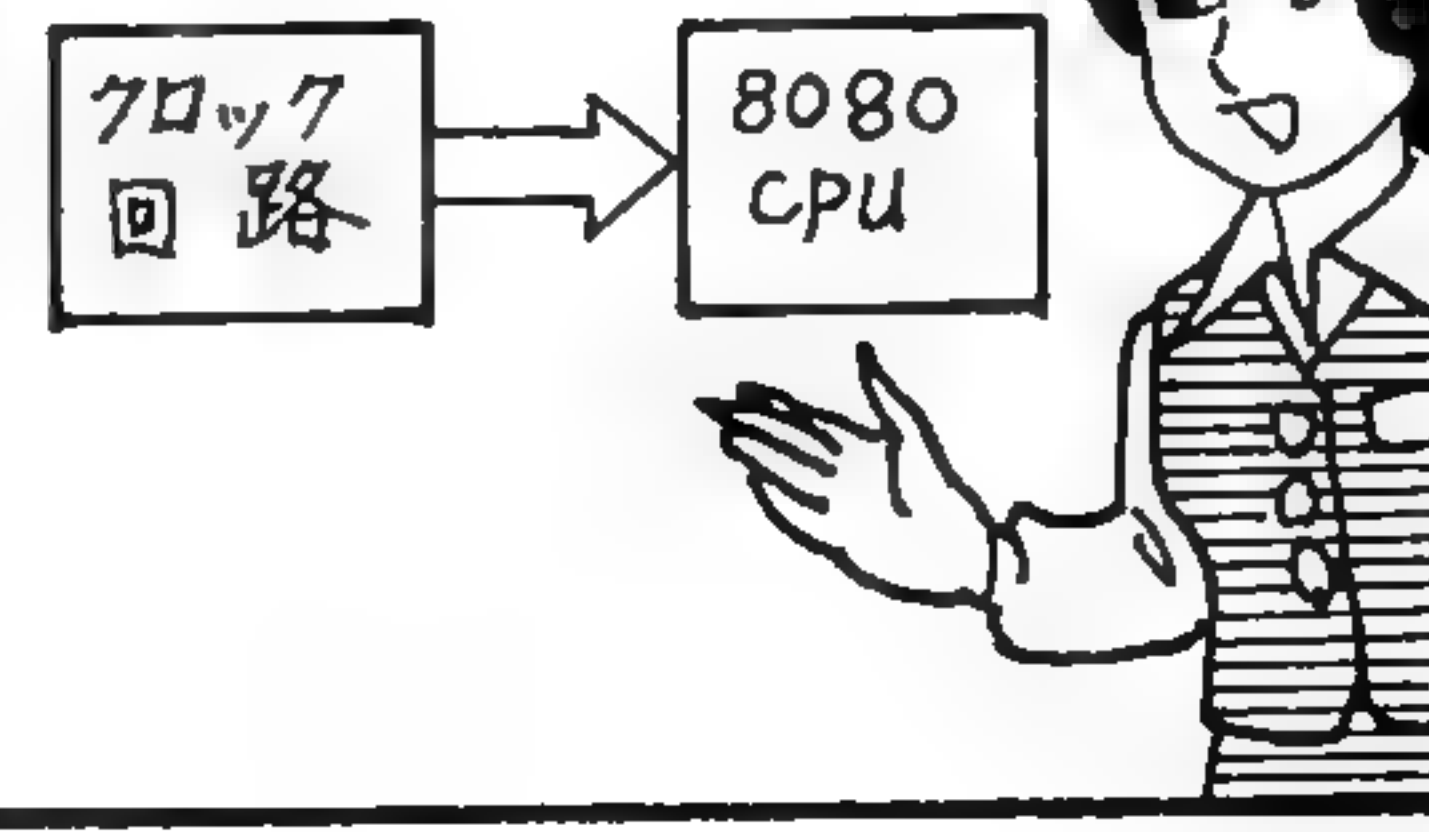
コンピュータはクロックで動作します。



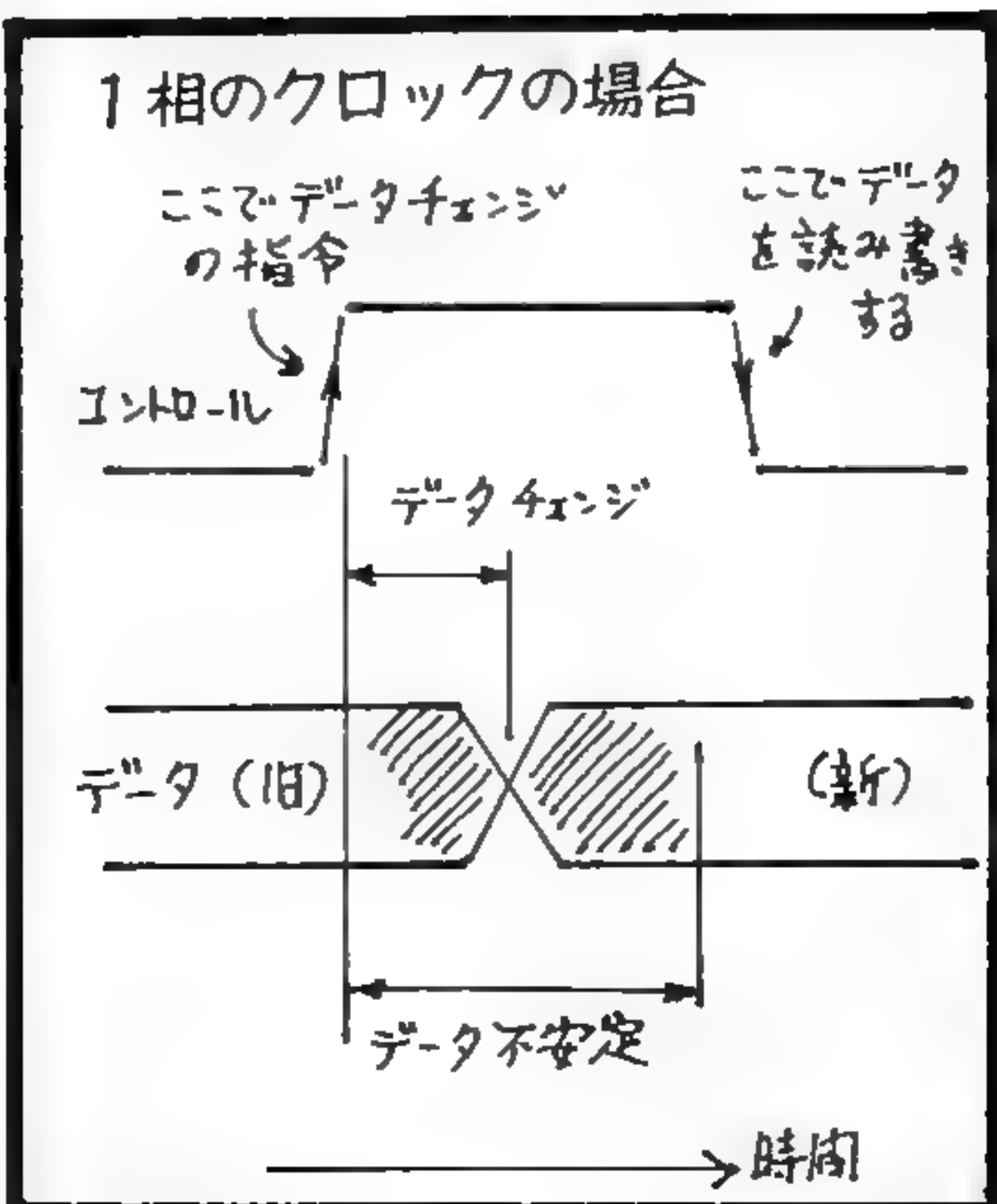
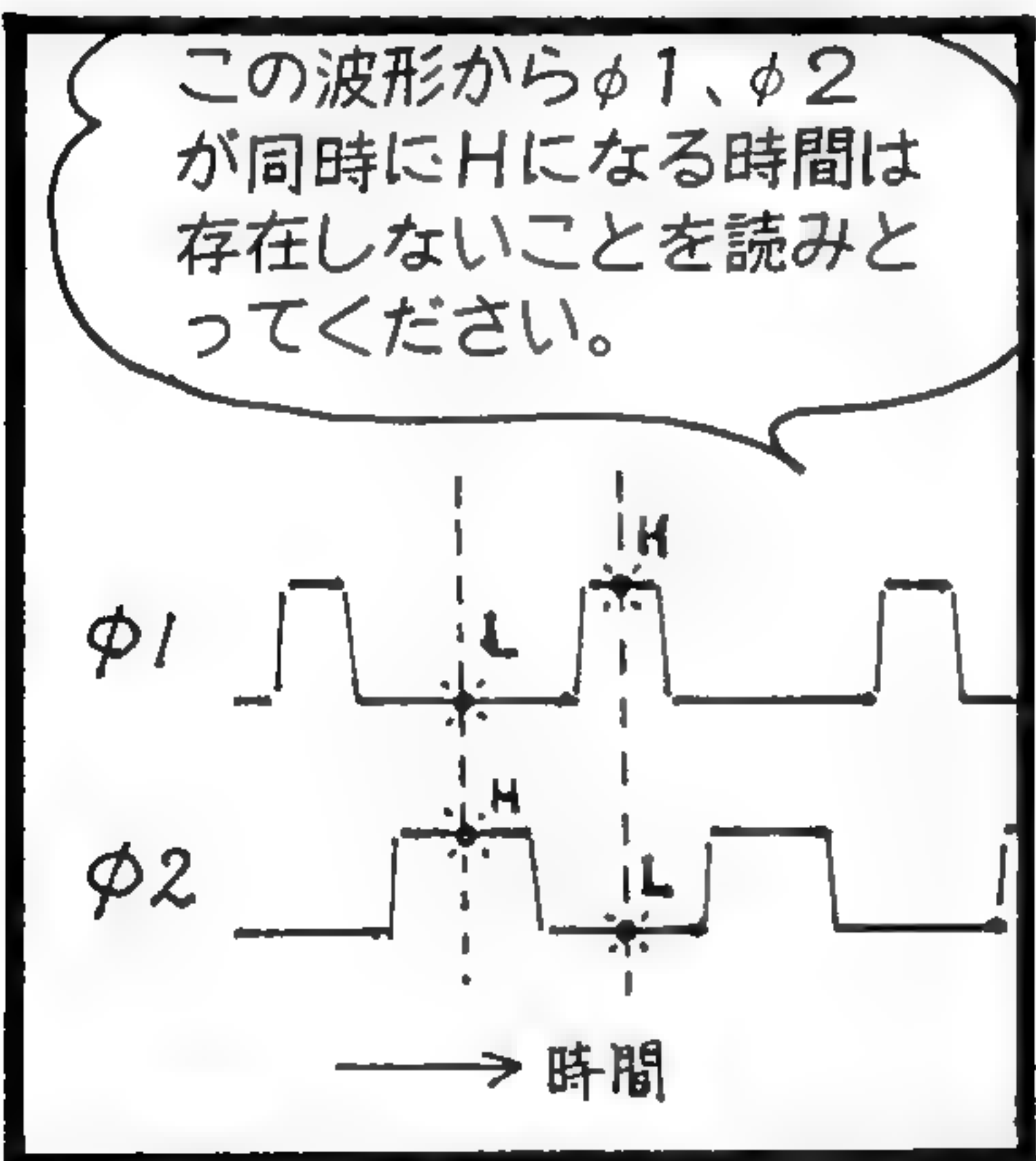
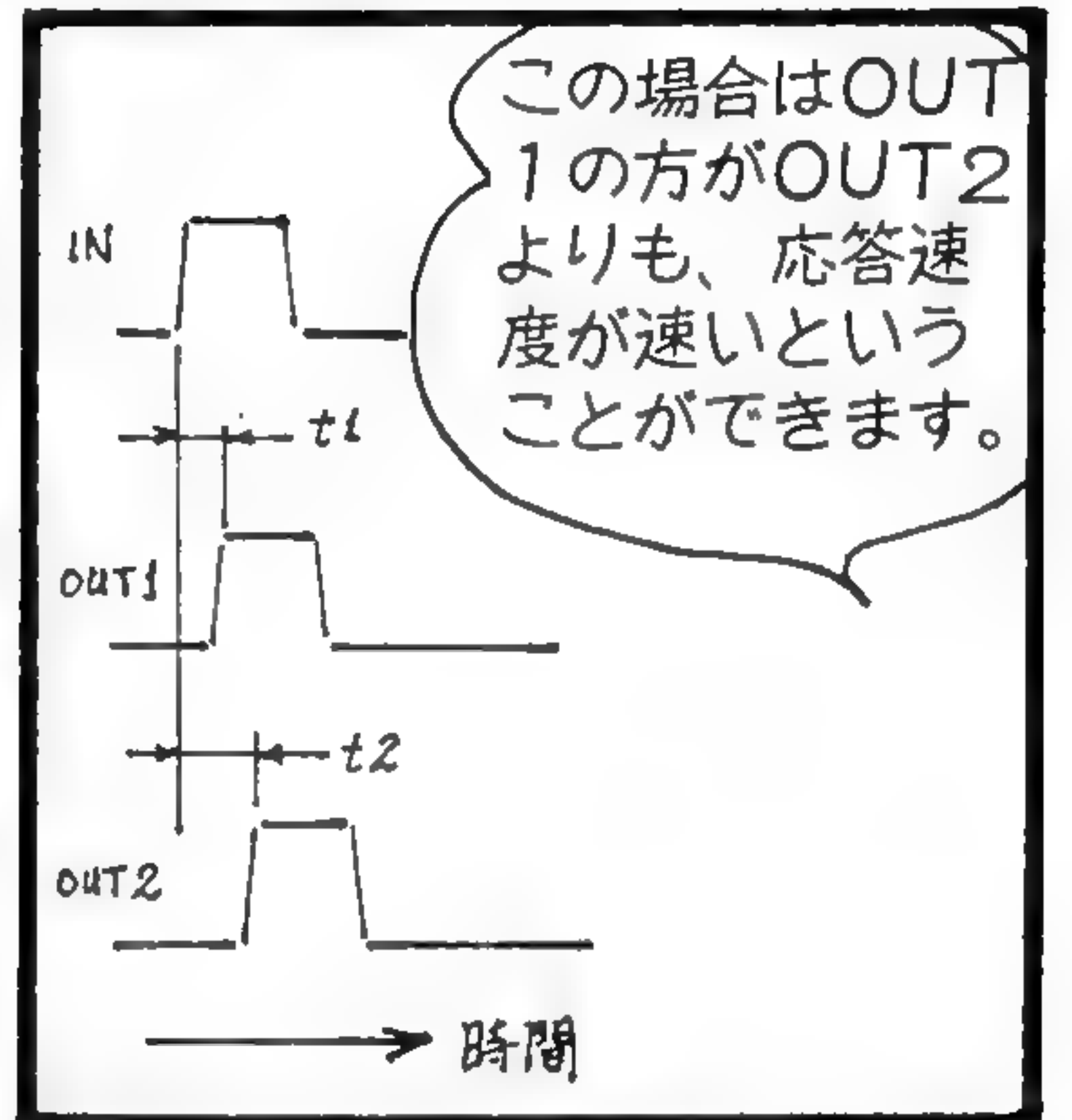
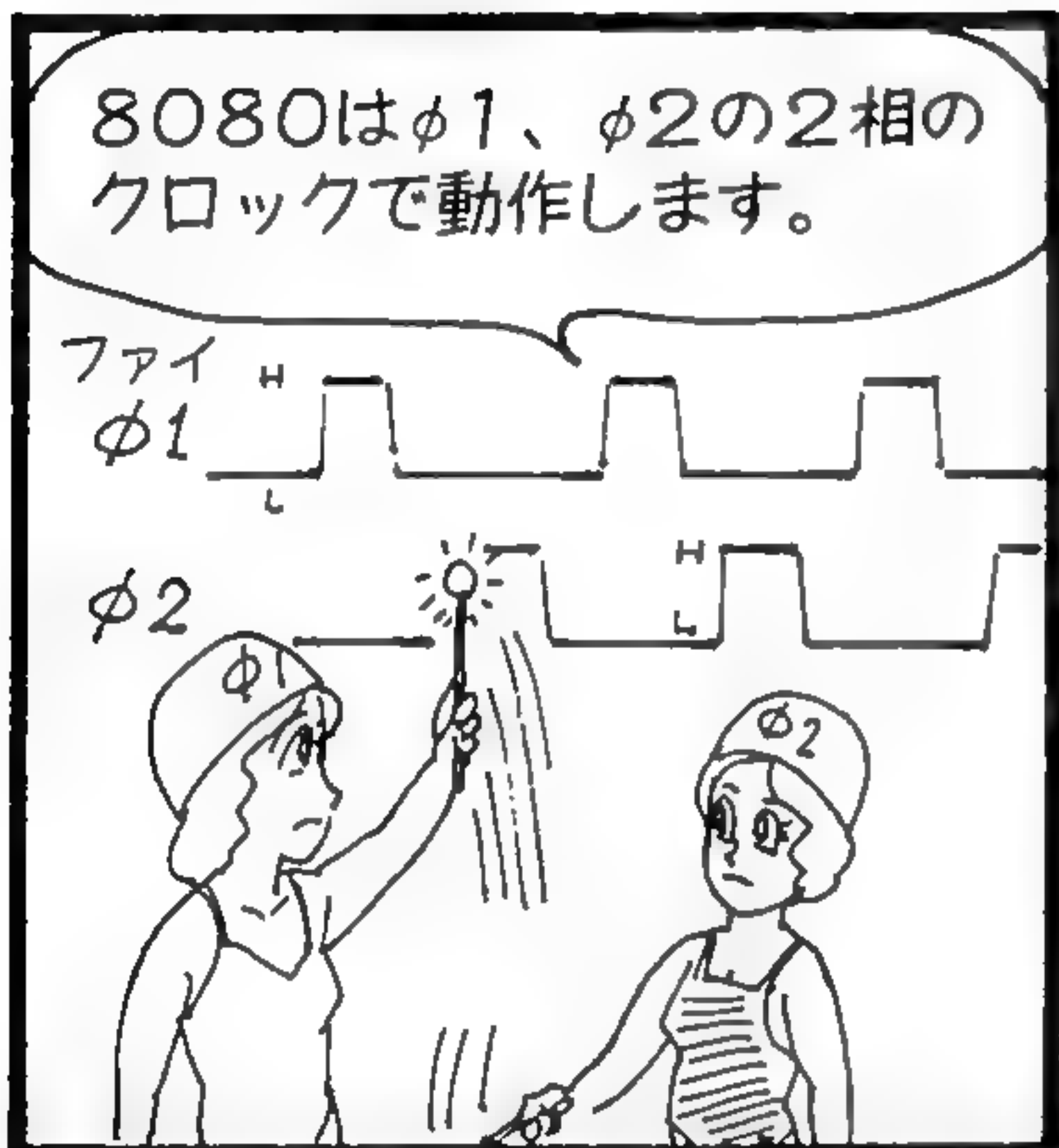
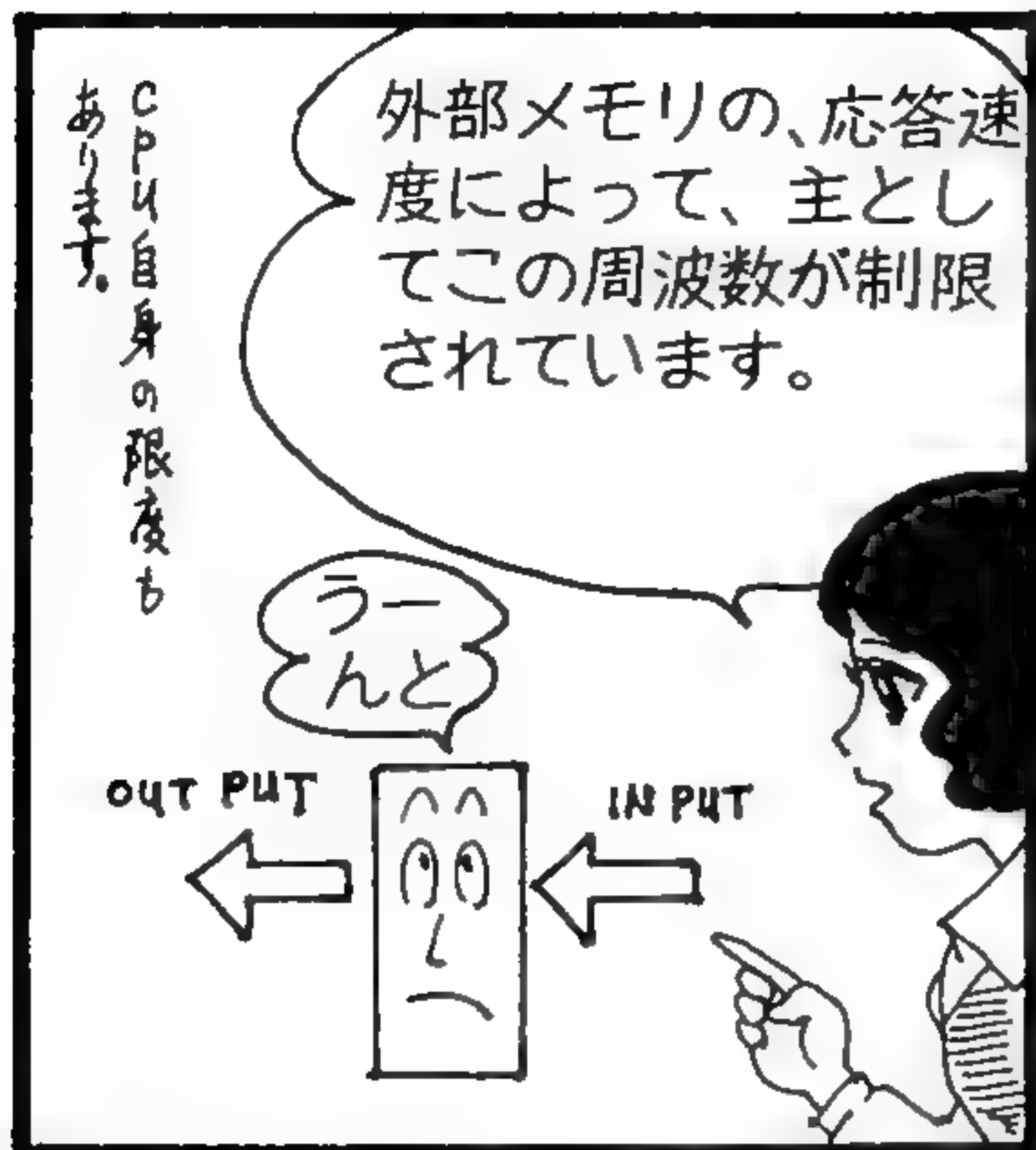
マイコンチップの内部では、クロックが入るたびに、何らかの動作をしているのです。プログラムがなくても動作しています。



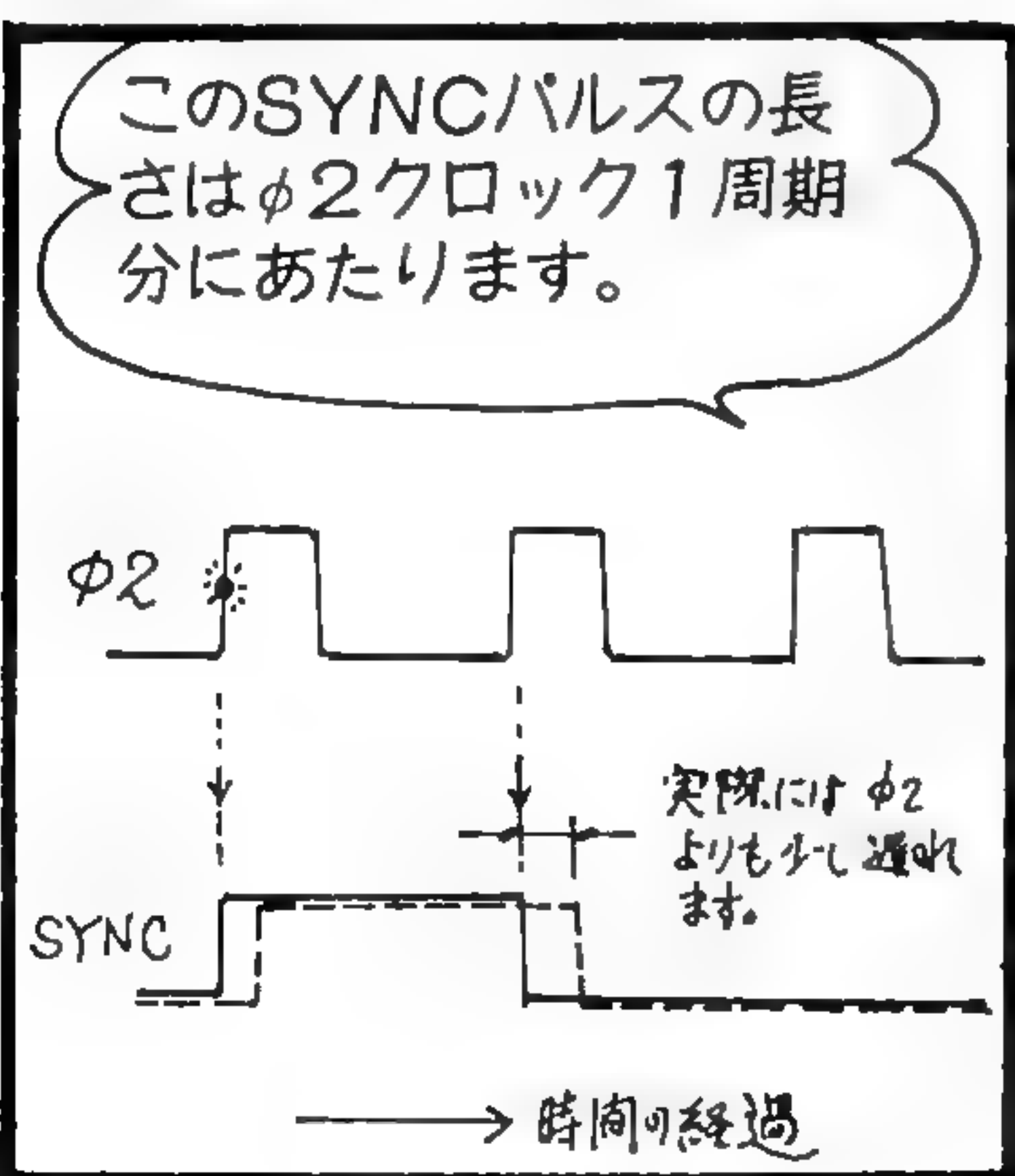
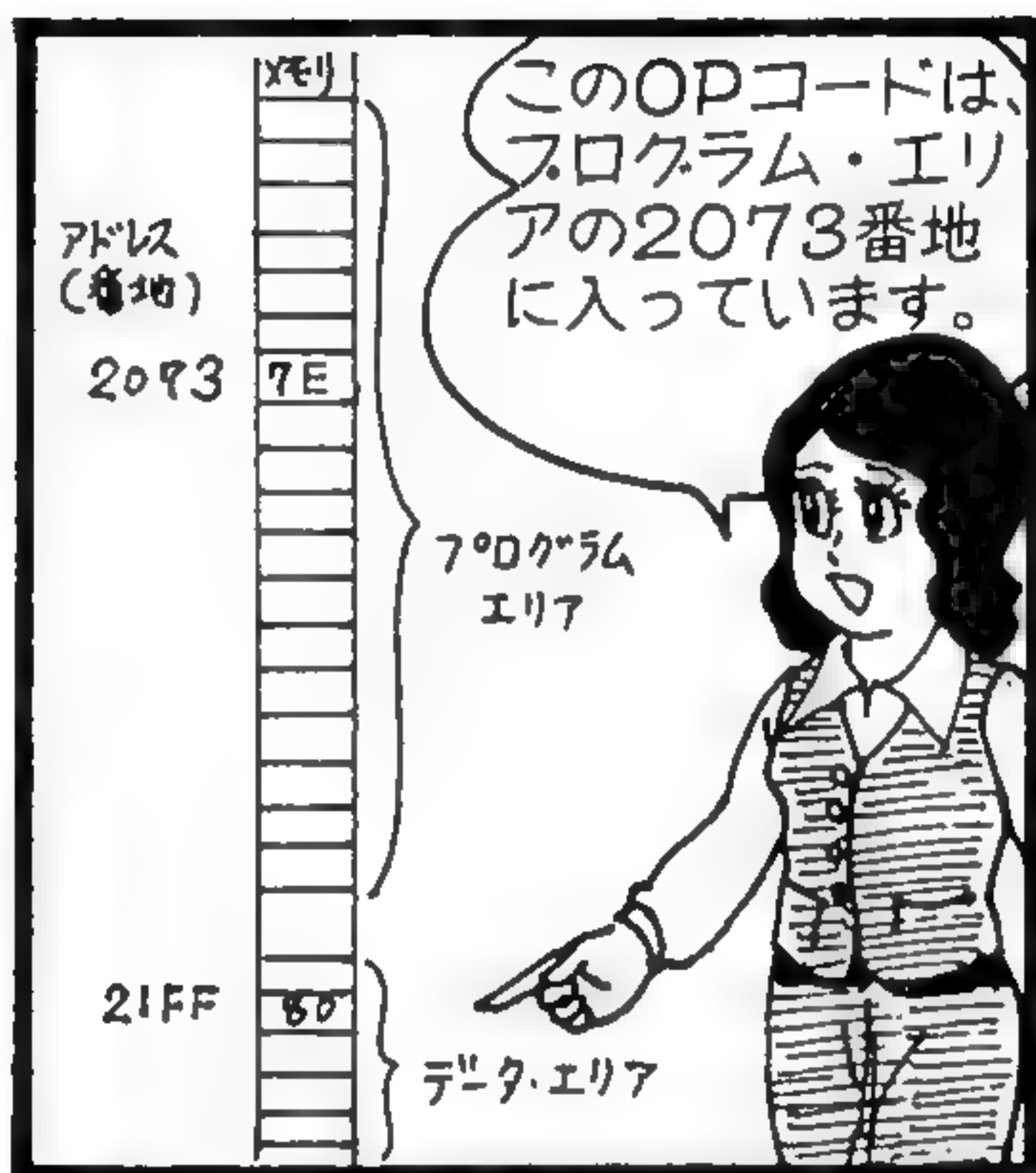
8080 CPUは、外部にクロック回路をつける必要がありますが、最近のものは、CPUチップに内蔵されているものもあります。



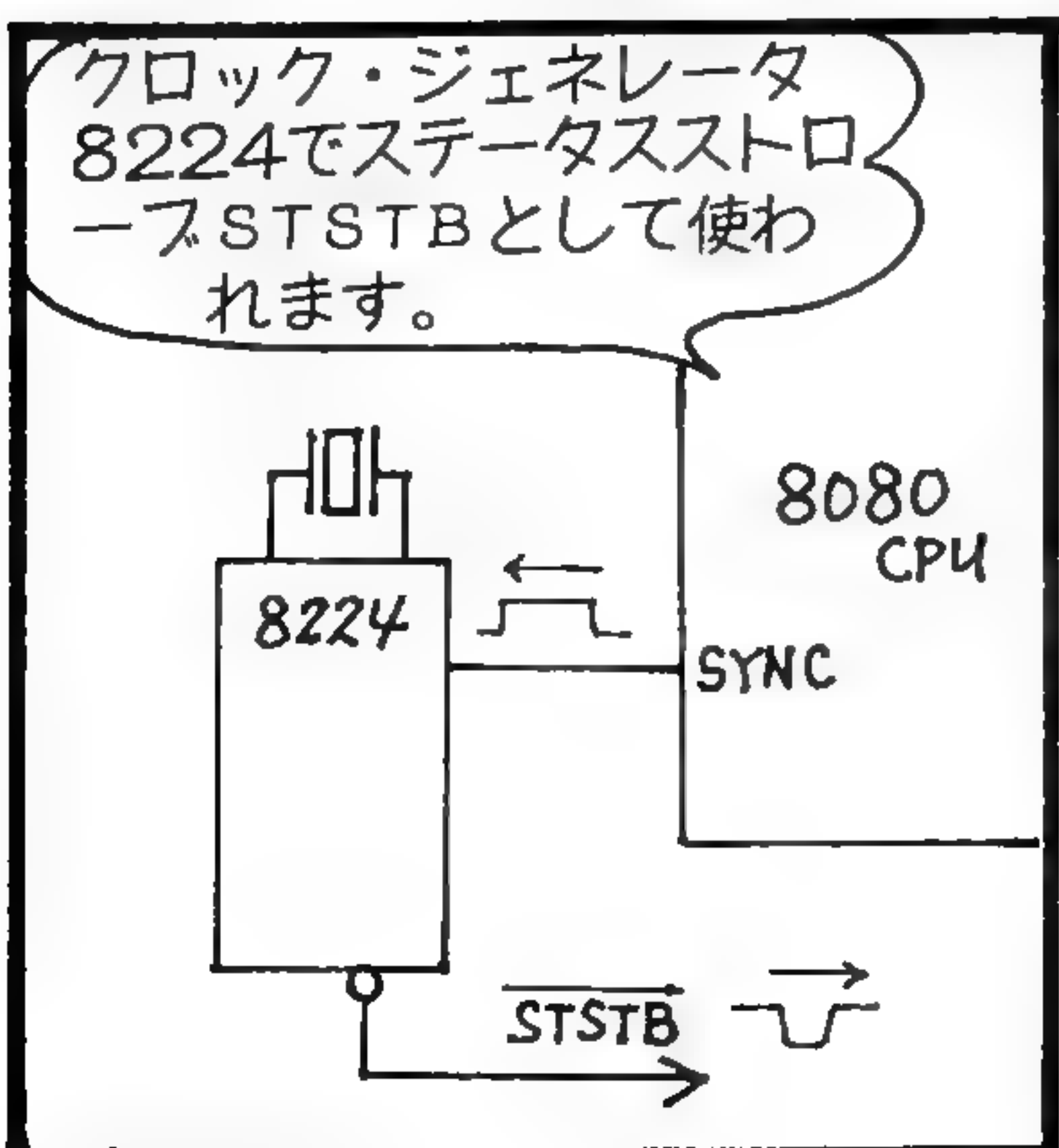
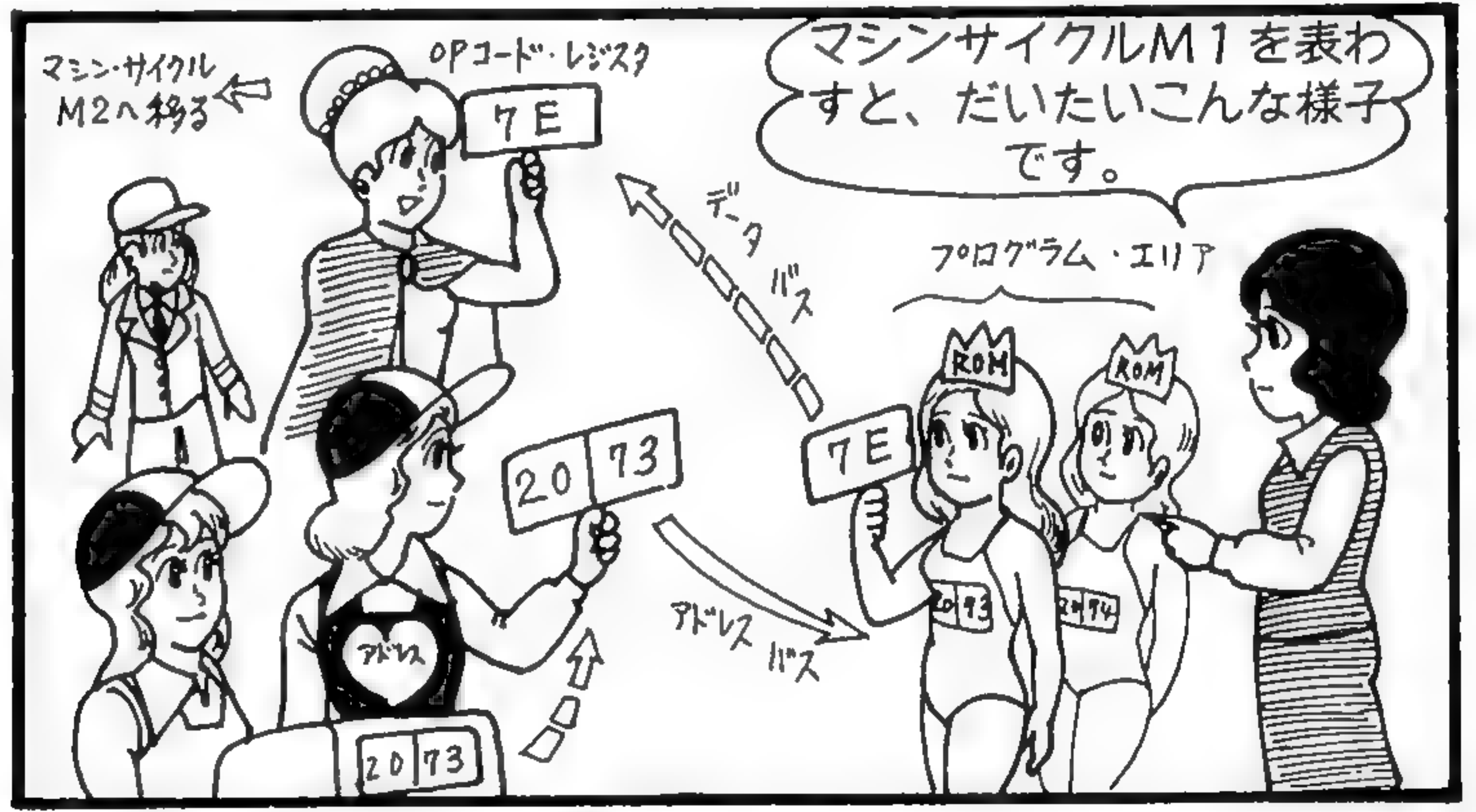
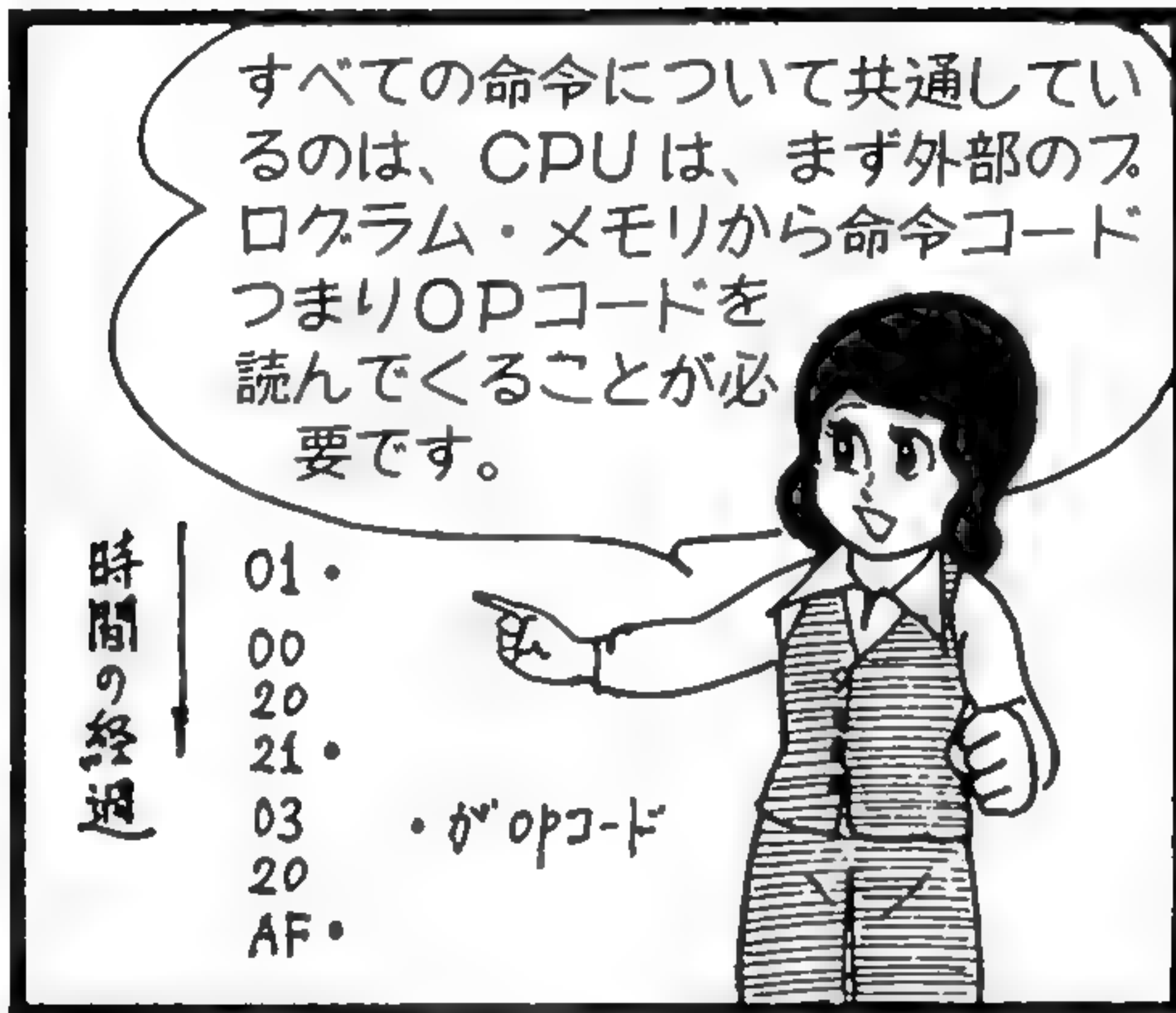
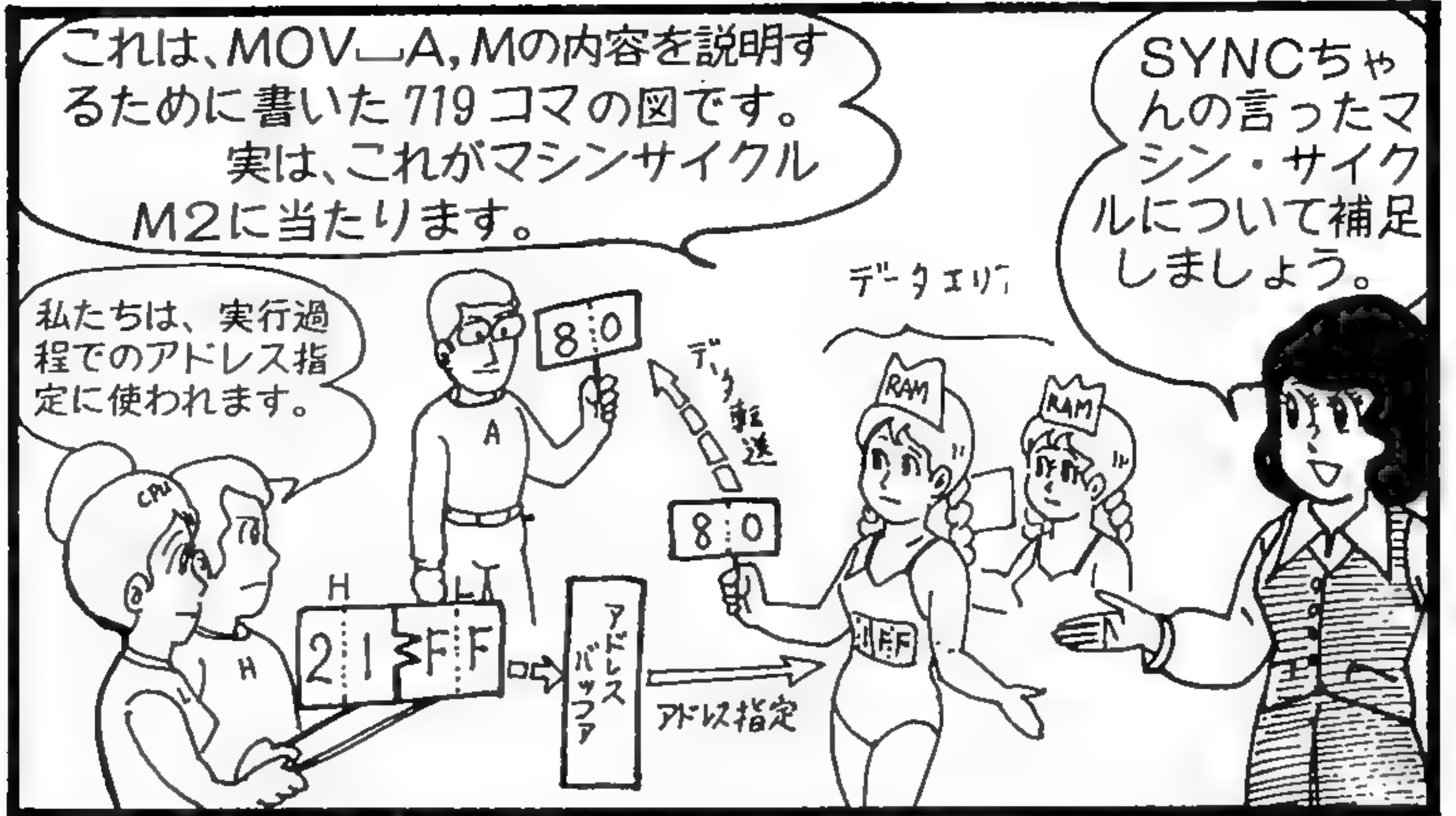






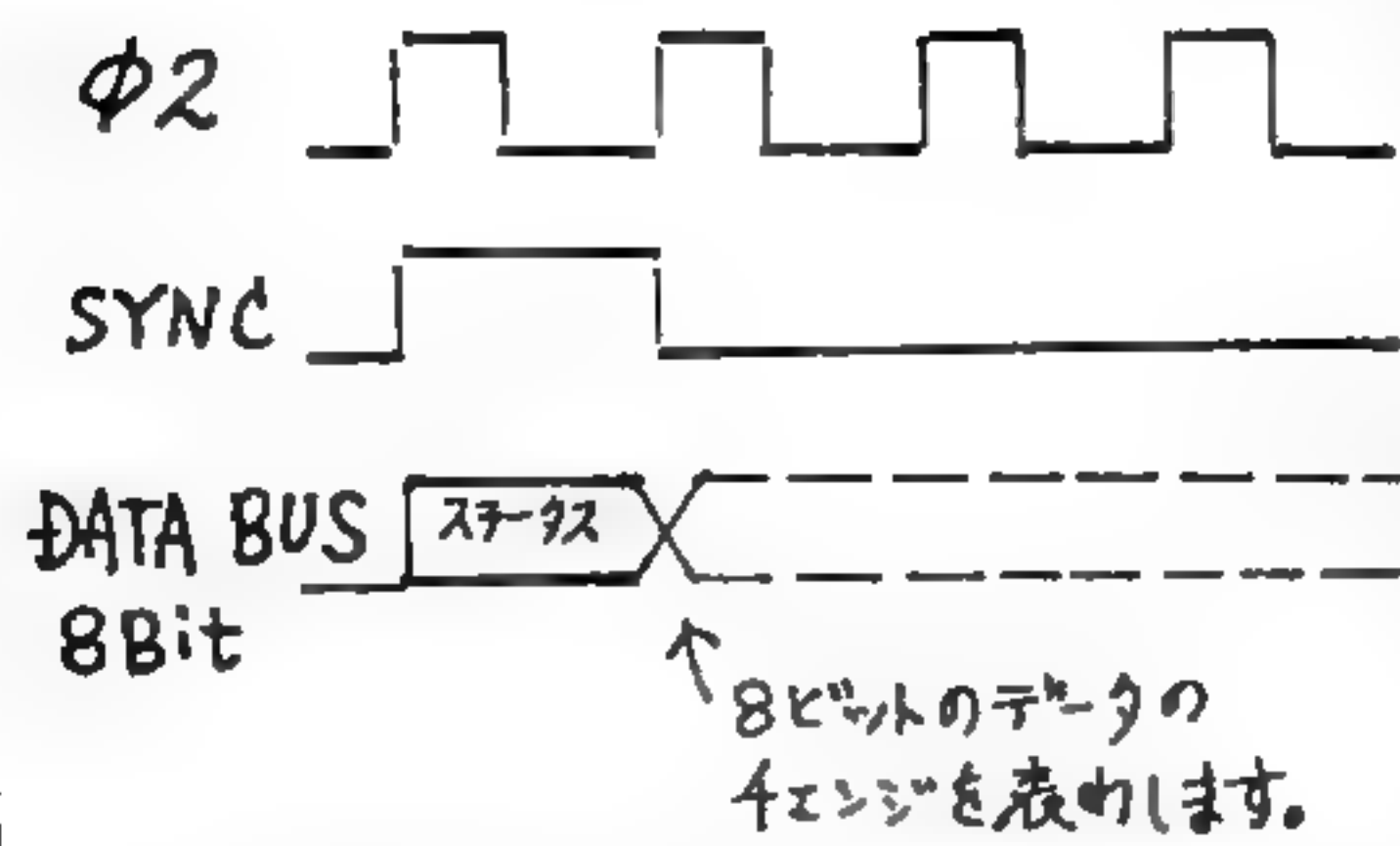








ステータスは、SYNCパルスと同じ時間にデータバスに現われます。



このステータス情報は、データバスからは、すぐに消えてしまうのでタイミングをとって外部回路でメモリする必要があります（データではないのでメモリとは言わず、ラッチすると表現します）。

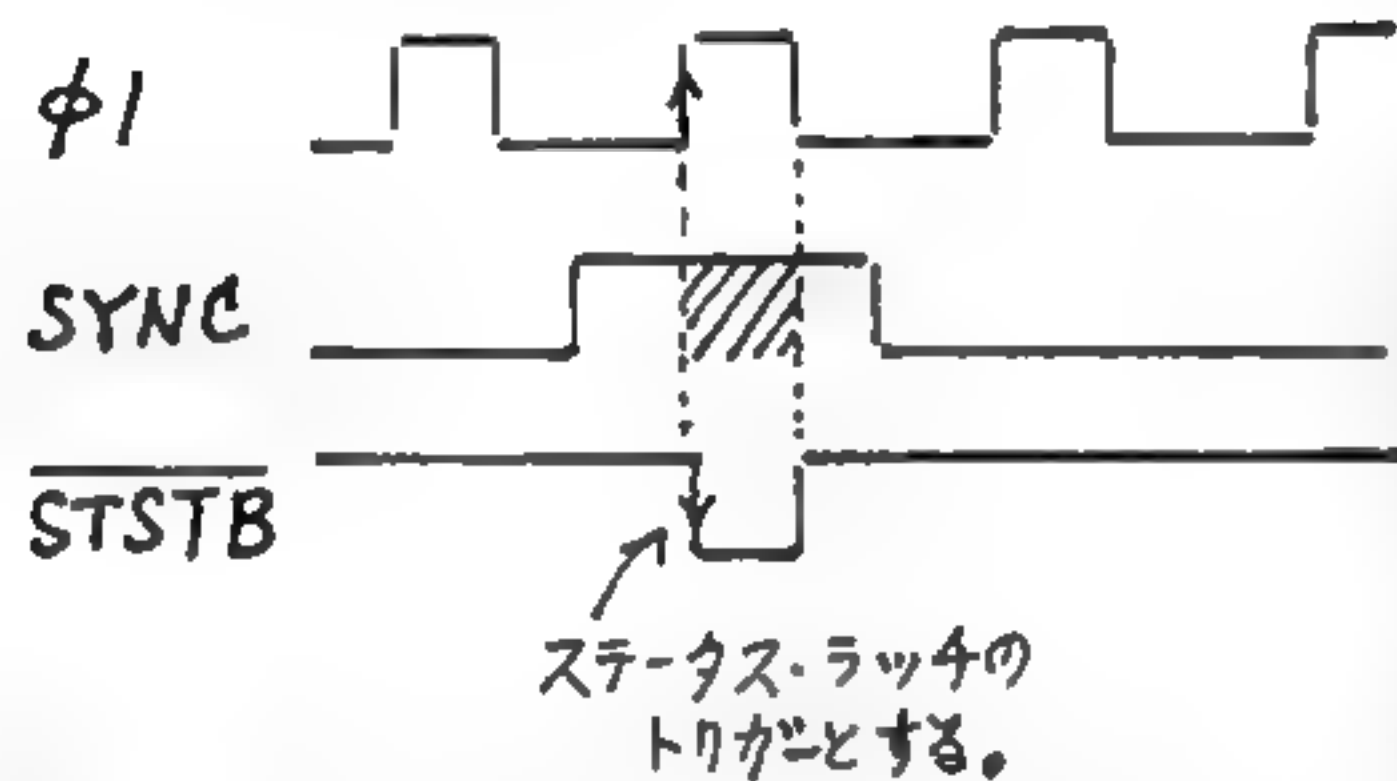
ラッチ  
電子的に  
鎖をかけて  
信号を  
保持する。



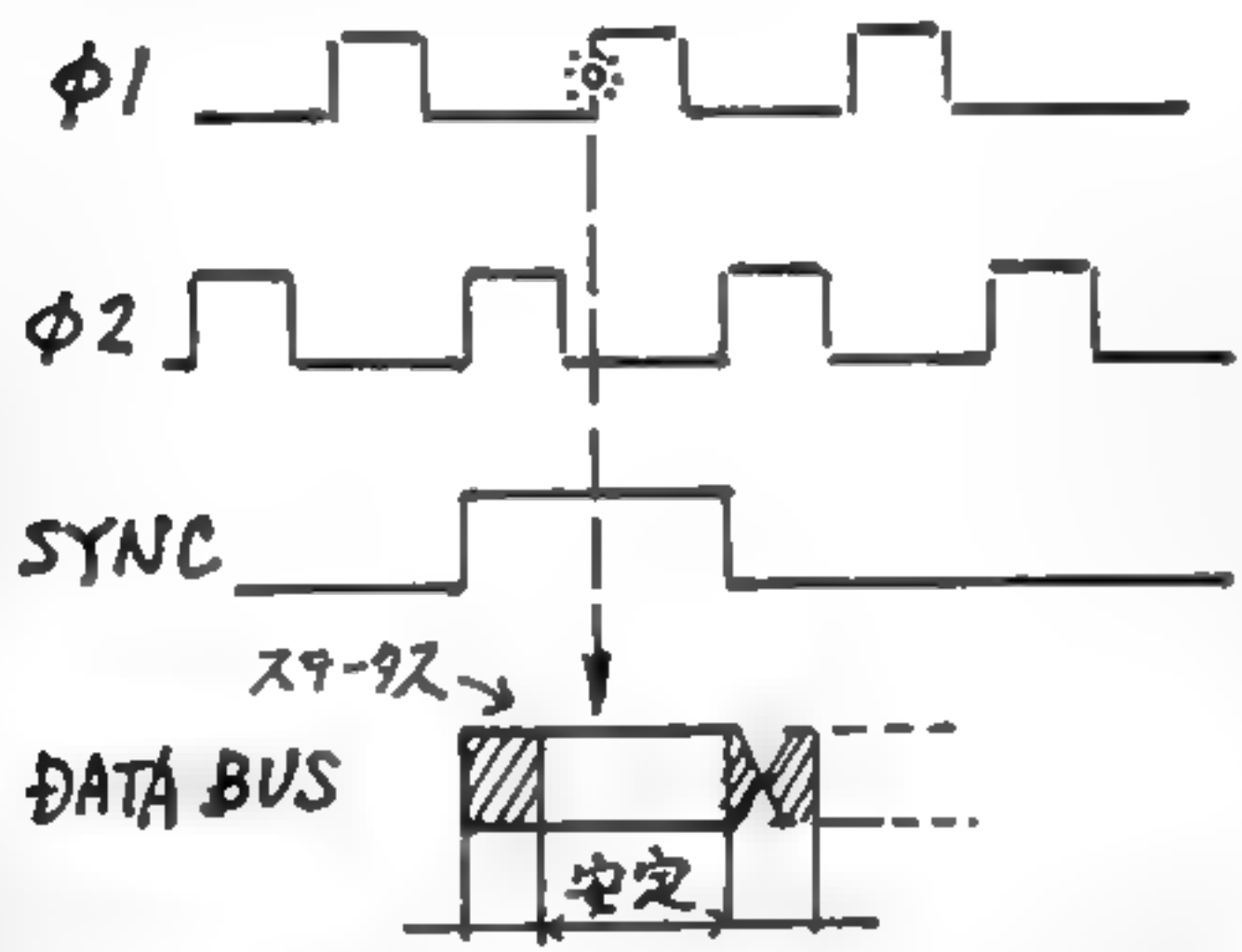
マシンサイクルの最初にCPUからはデータバス8ビットにステータス情報が出力されます。



ここで、φ1に注目すると、φ1のエッジはデータの安定域にありますね。φ1は次々にくるので、SYNCとANDをとってSTSTBとして、これをトリガーとします。



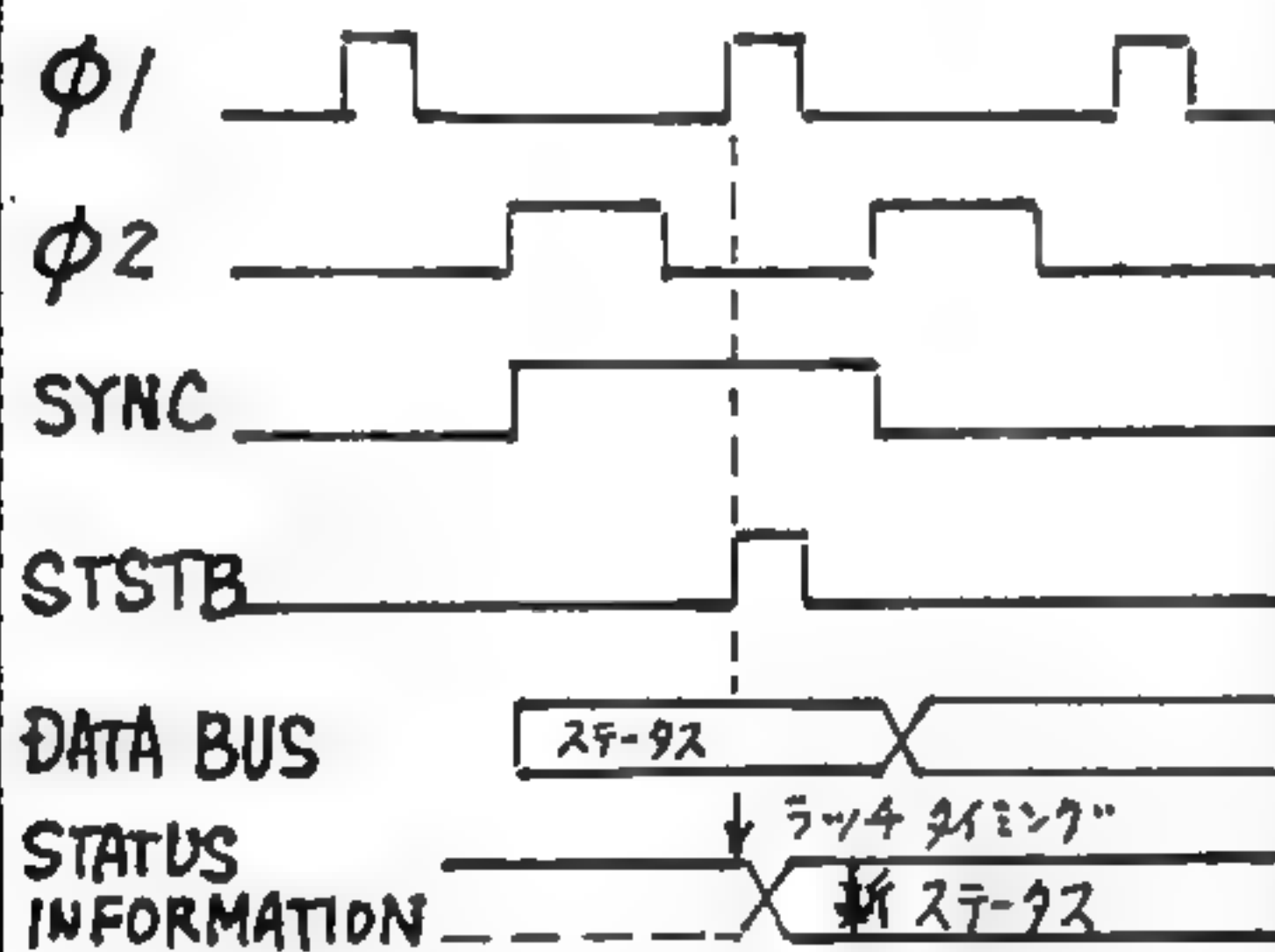
正確にデータをラッチするためにはパルスの両端の不安定な時間帯でのトリガーを避けなければなりません。



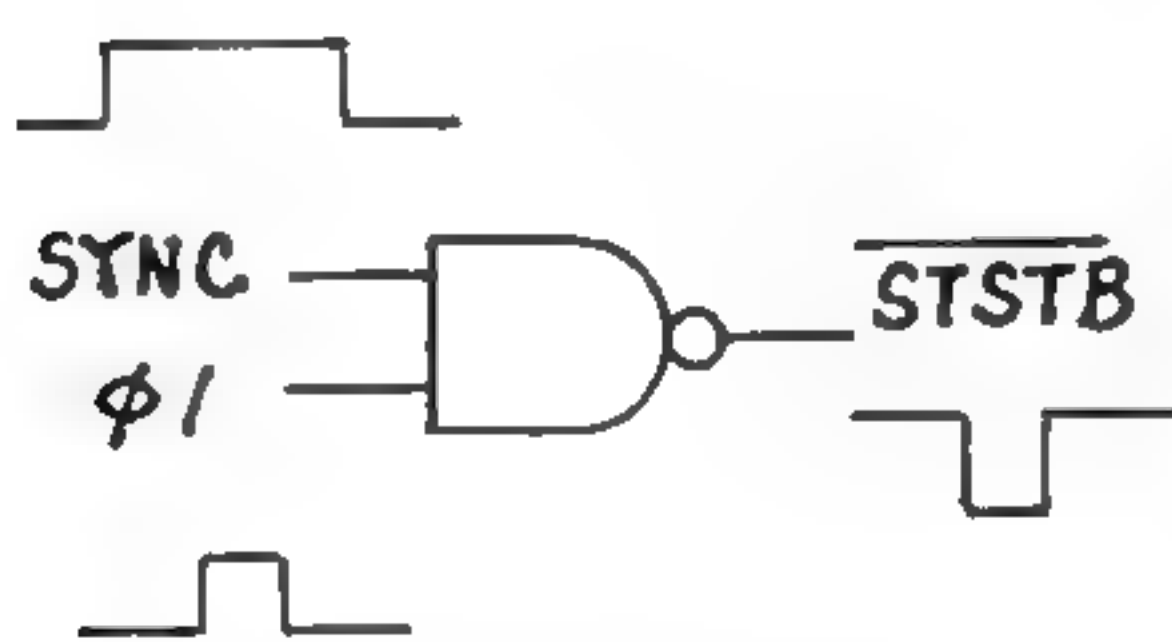
データバスは8ビットありますから、各ビットごとの応答時間にはバラツキがありエッジの部分の波形はみだれています。



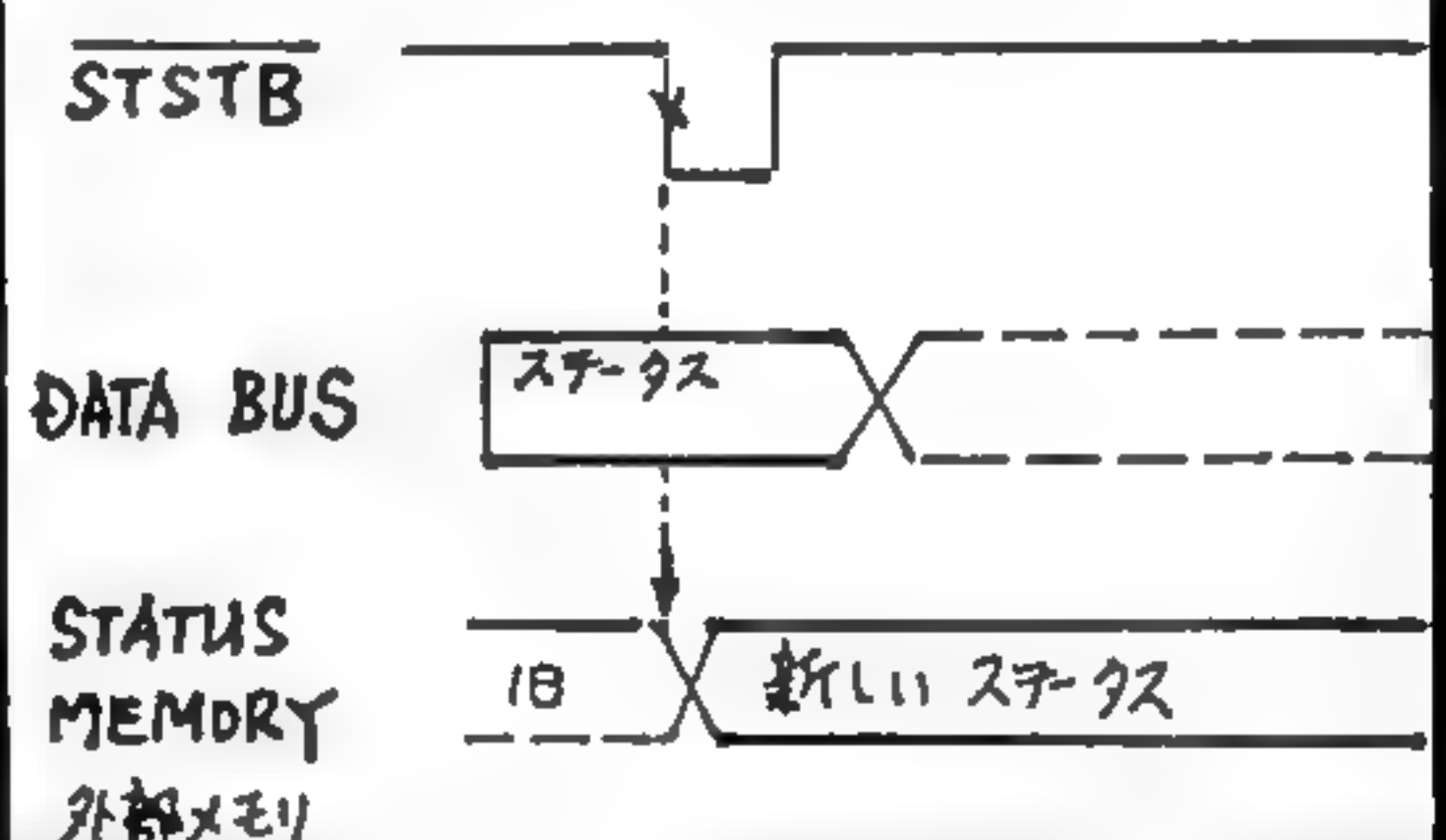
教科書にはこれらのことが、ひとつのタイミングチャートに表わされています。



STSTBは、STATUS STROBEの略です。これと、SYNC、φ1との関係はNANDゲートのインプットとアウトプットになっています。



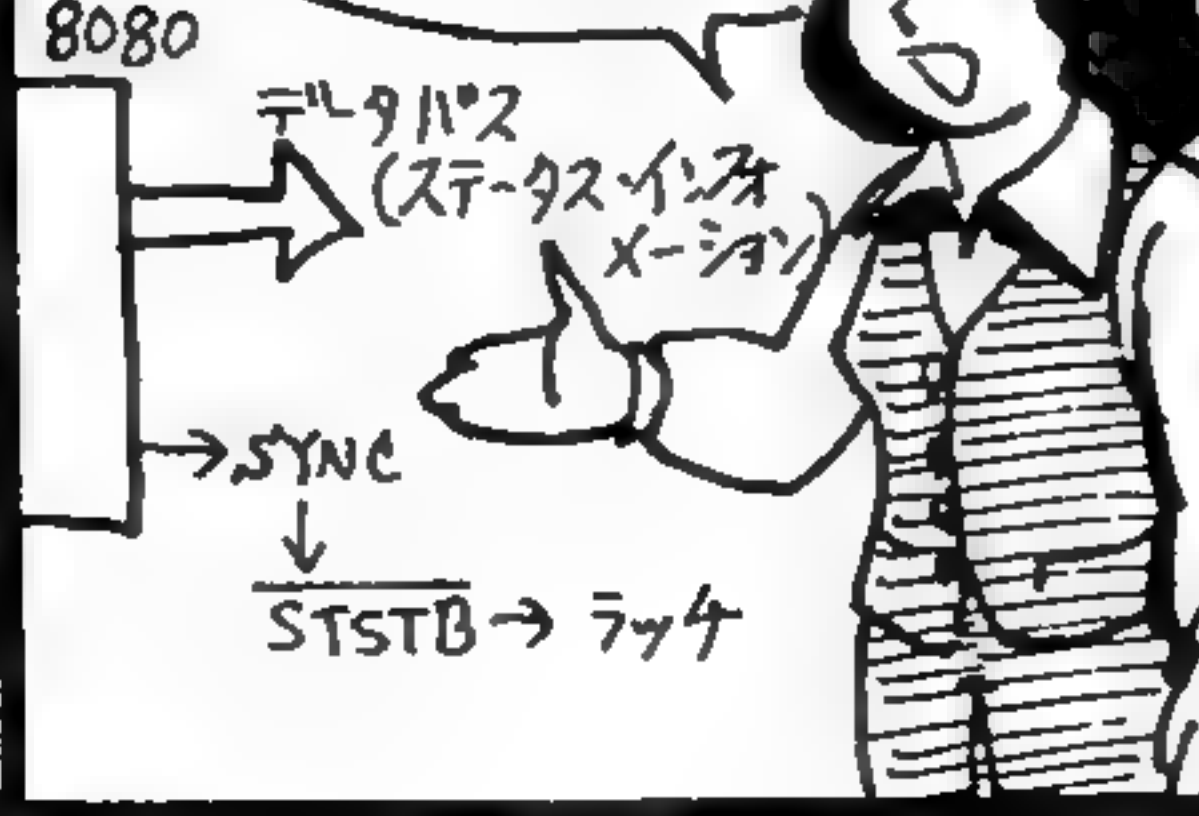
このSTSTBとタイミングをとって、外部メモリにデータバスの内容をラッチすると、それがステータス情報として、次に送られるデータの処理に使われるわけです。



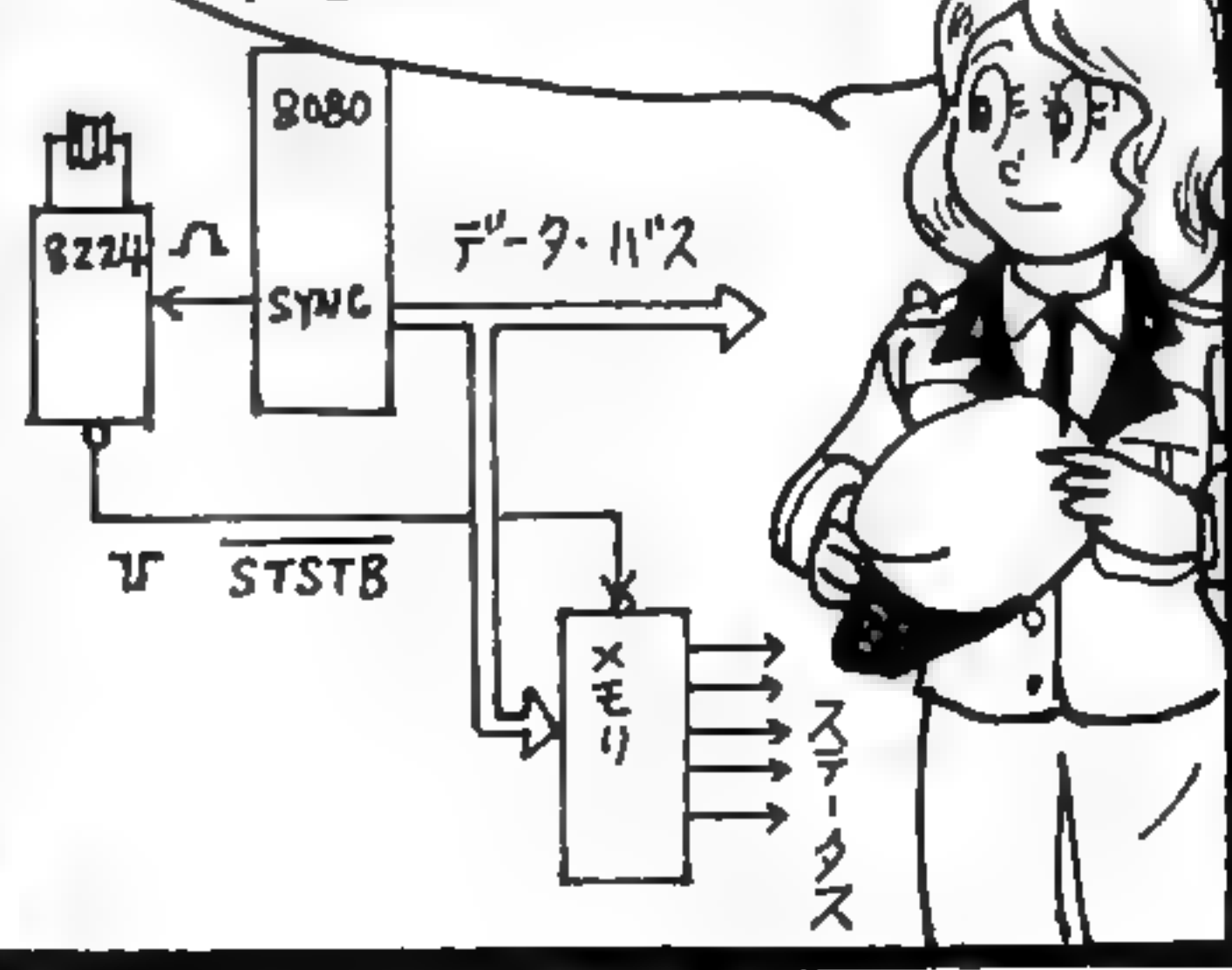
これからはS.Iと略します。このS.IはCPUが次のマシンサイクルで何をするつもりなのか外部に知らせるためのものなのです。



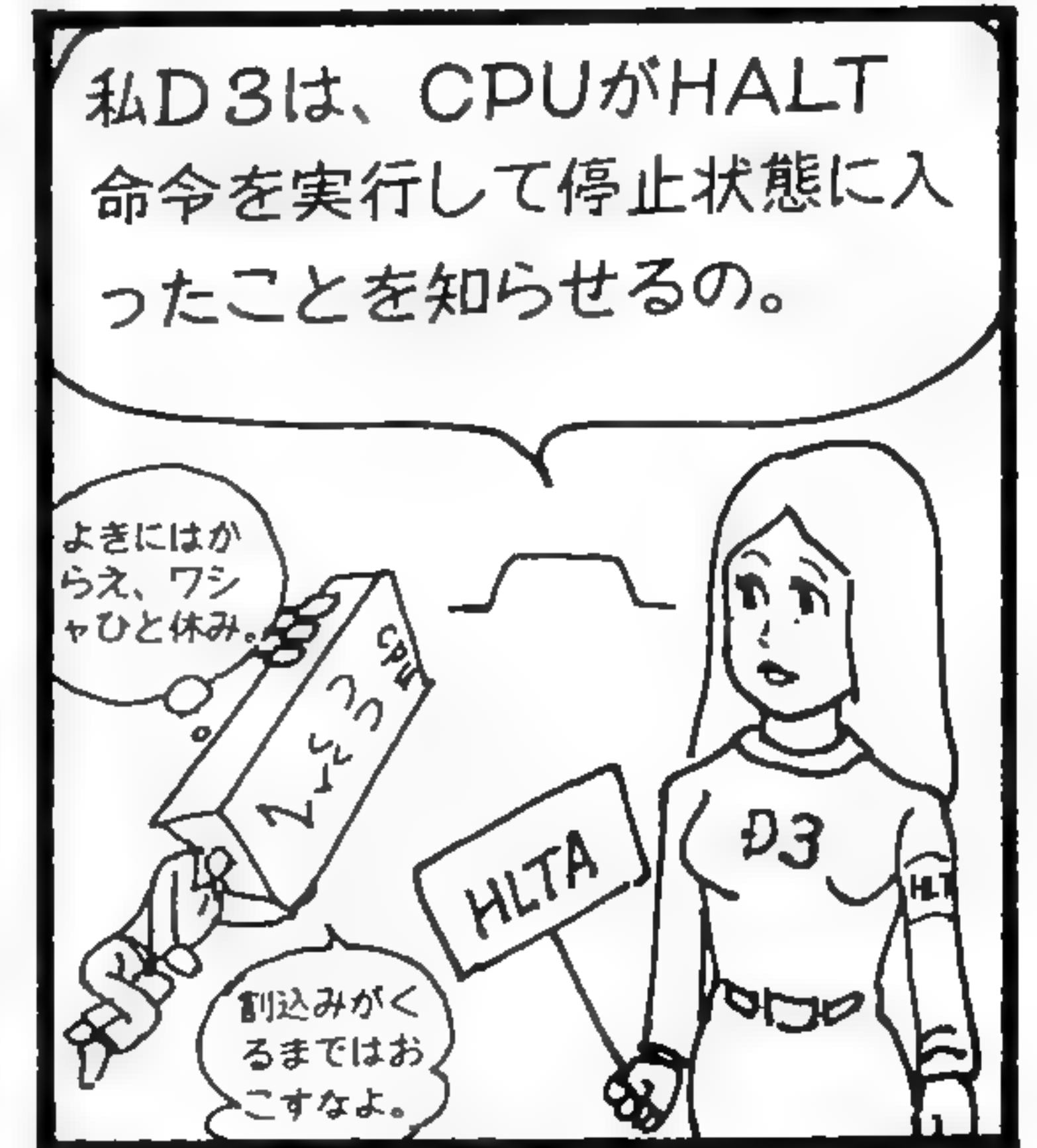
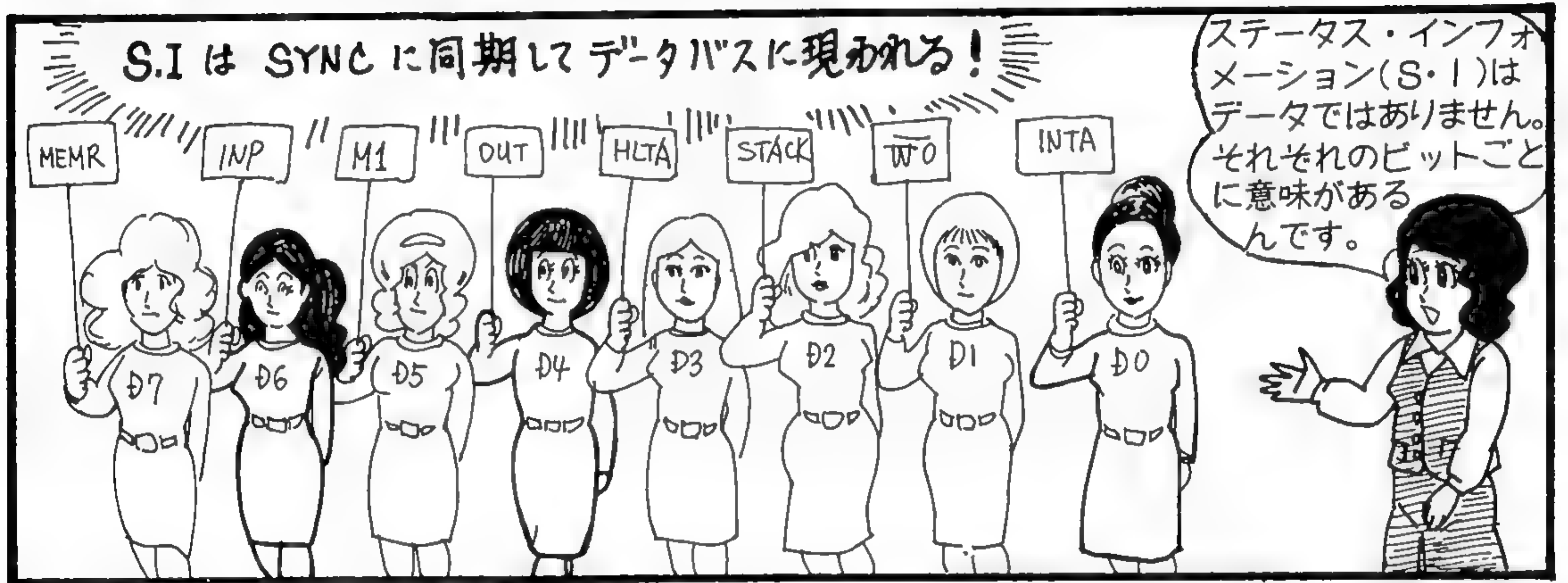
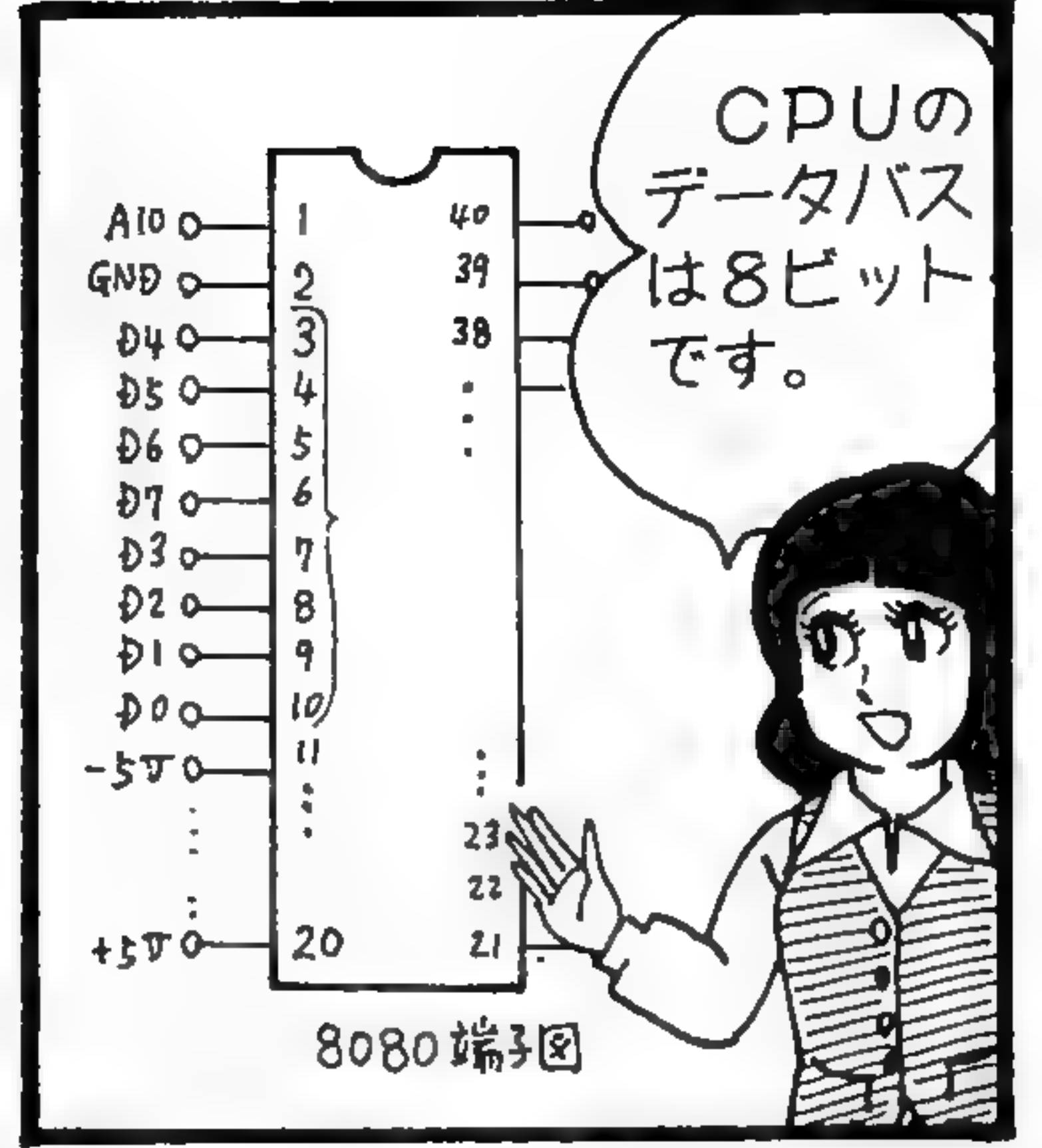
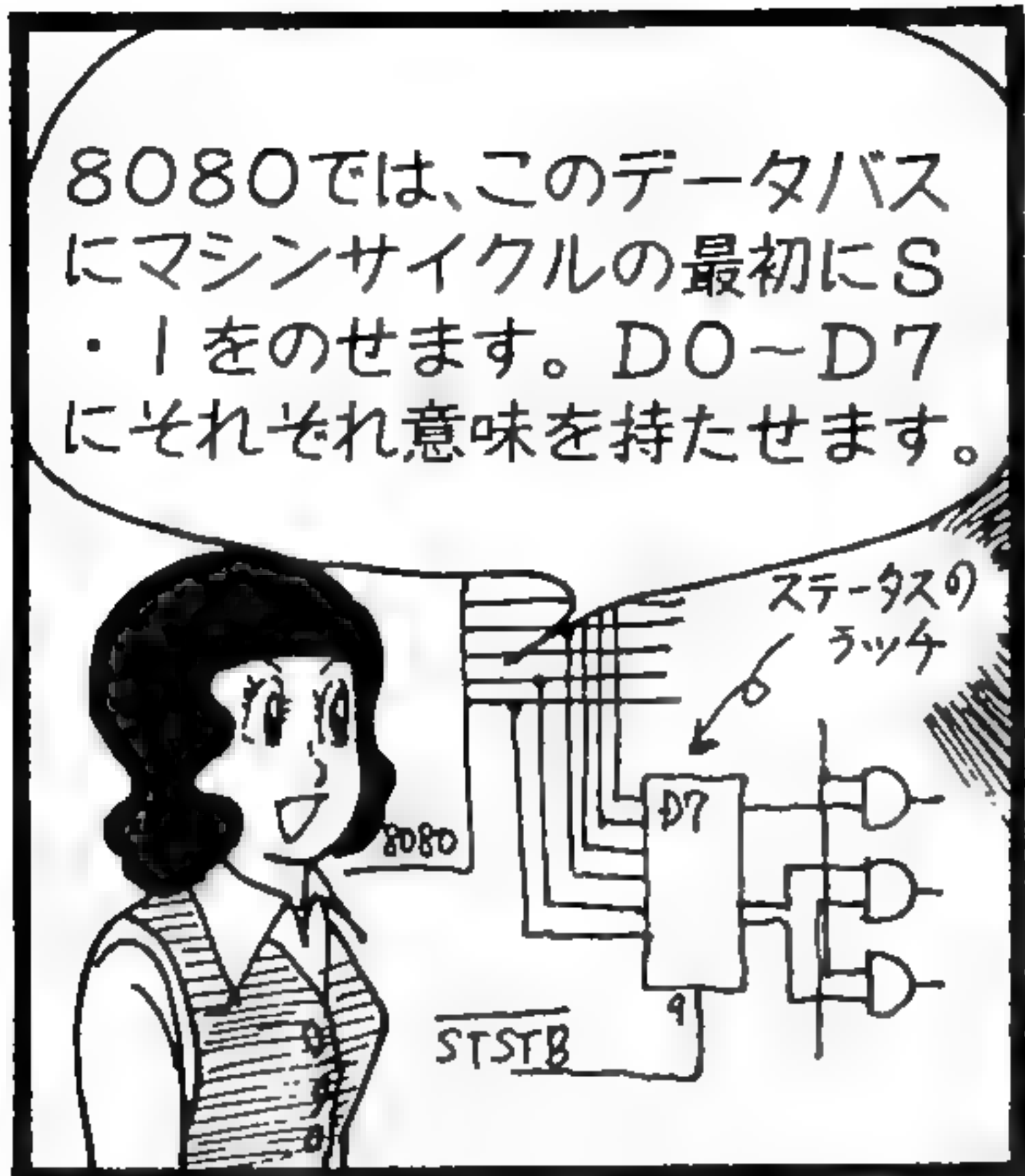
8080ではSYNCのタイミングでデータバスにステータス・インフォメーションというものがのっている、ということです。



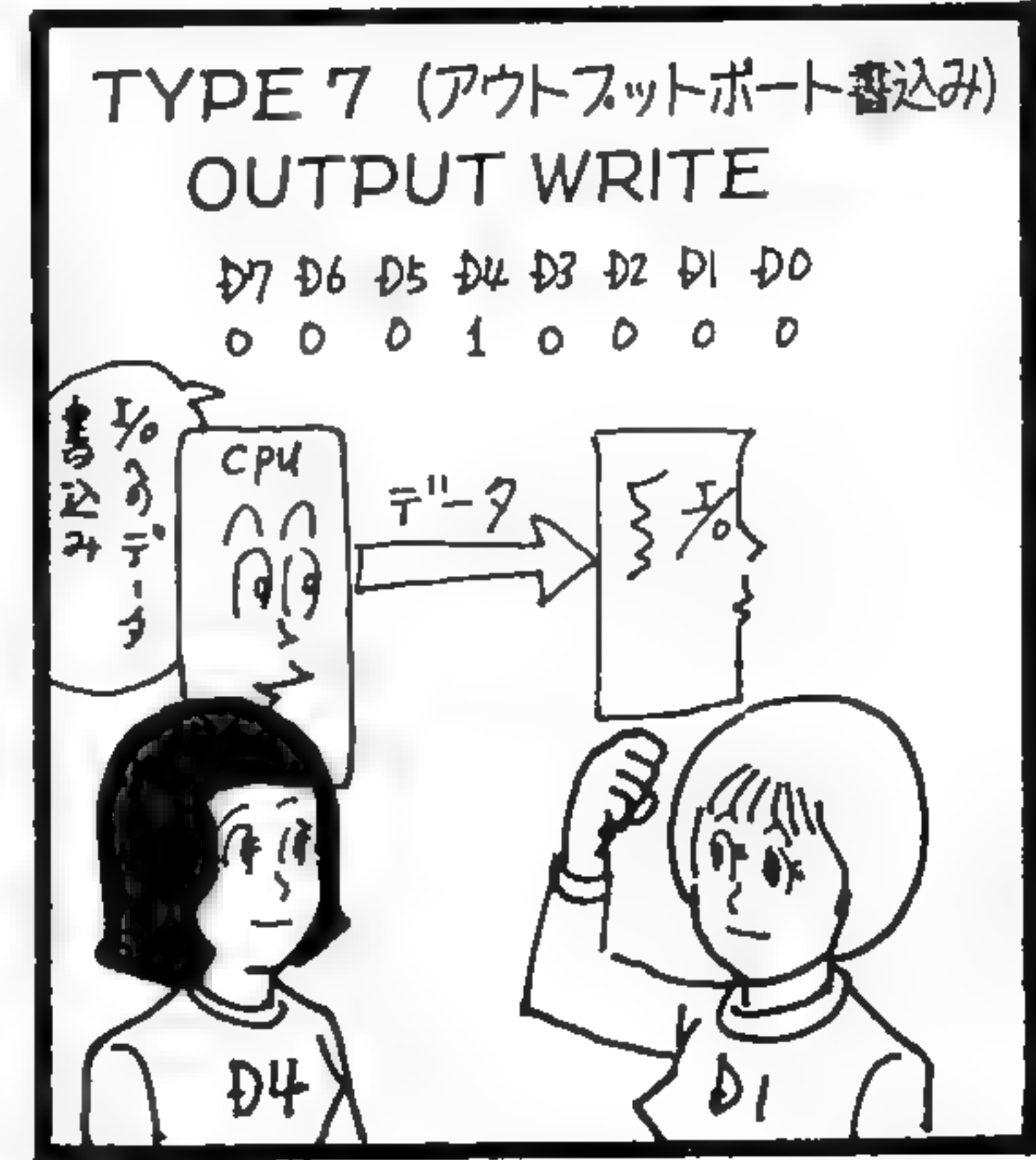
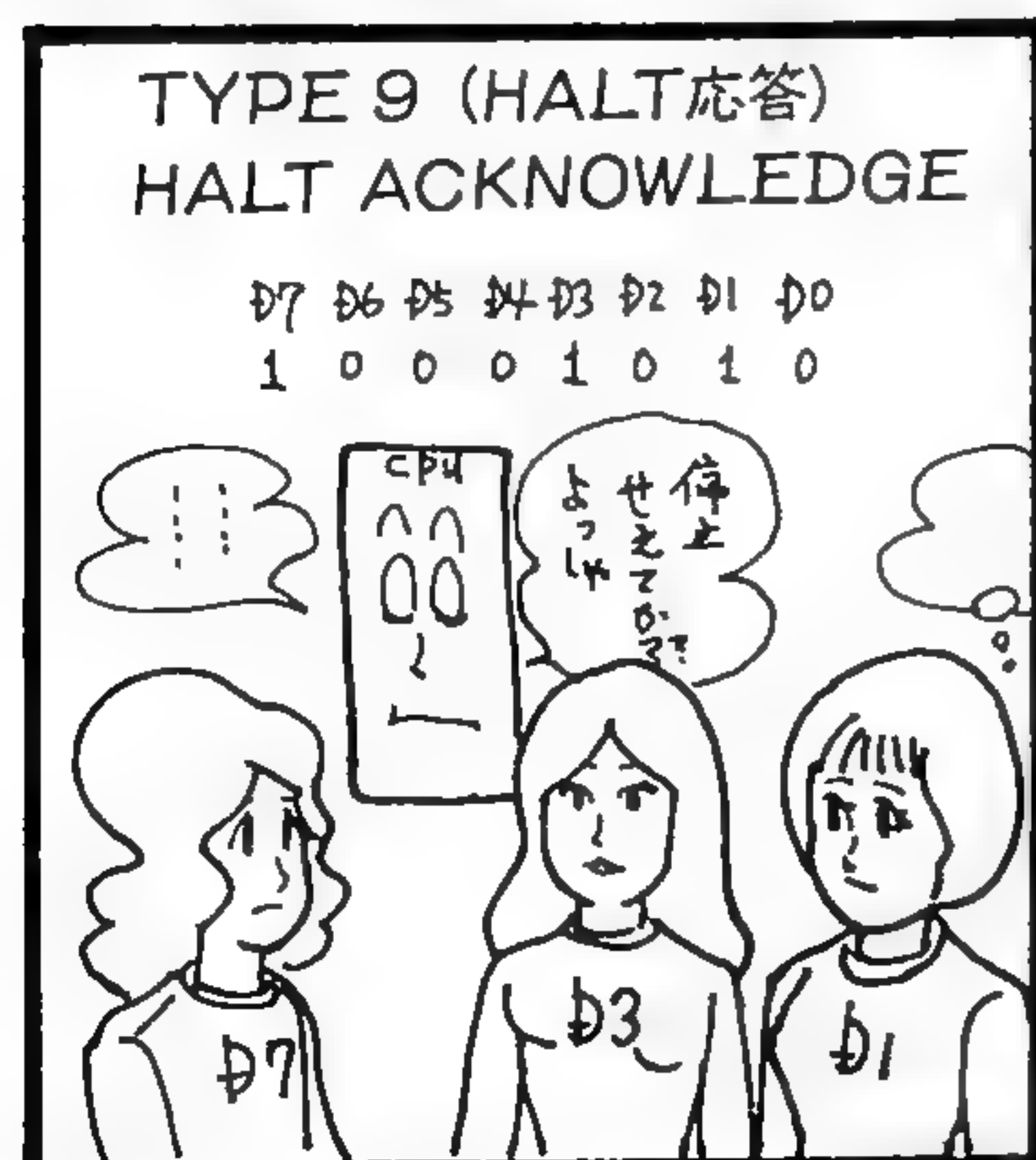
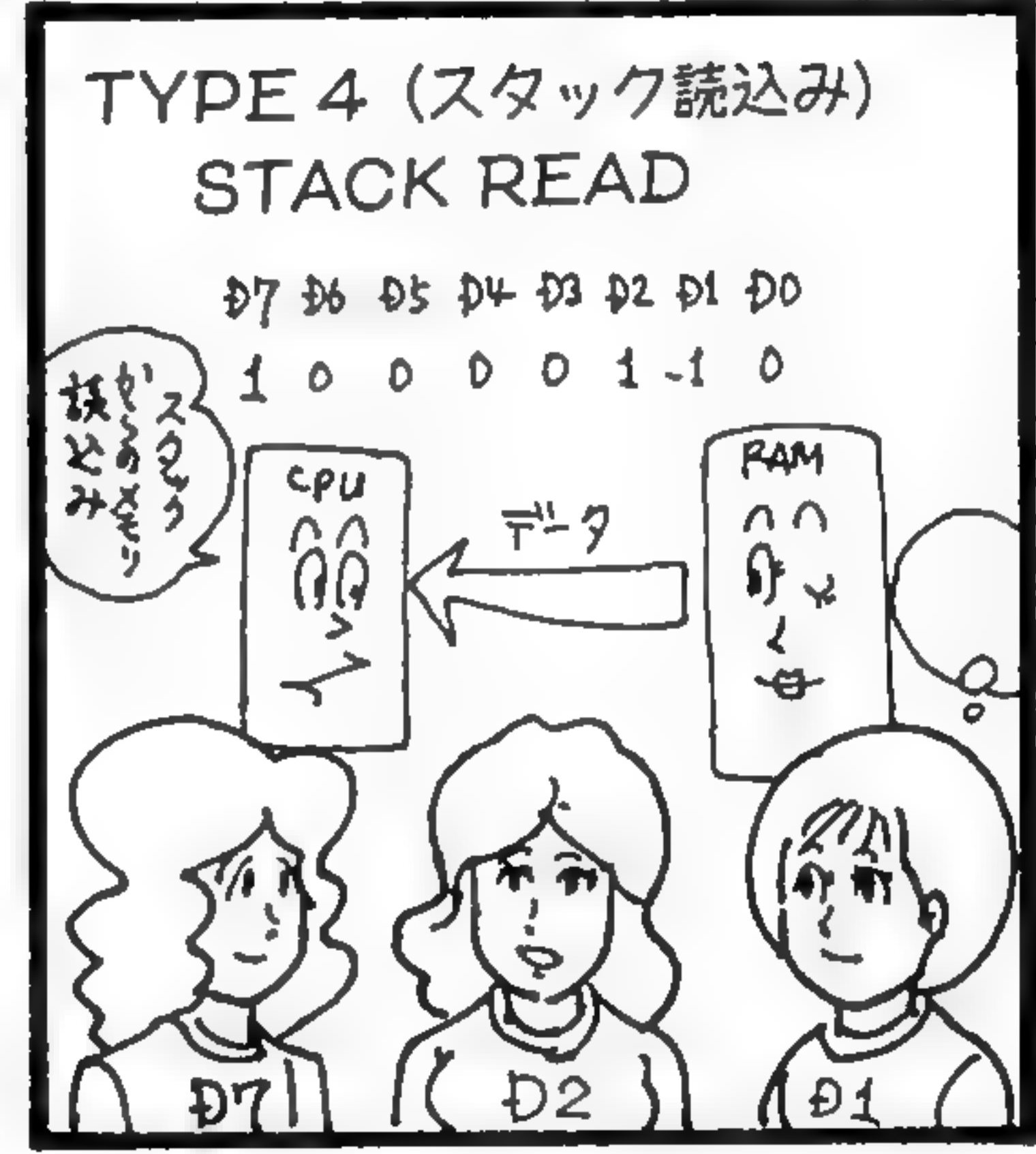
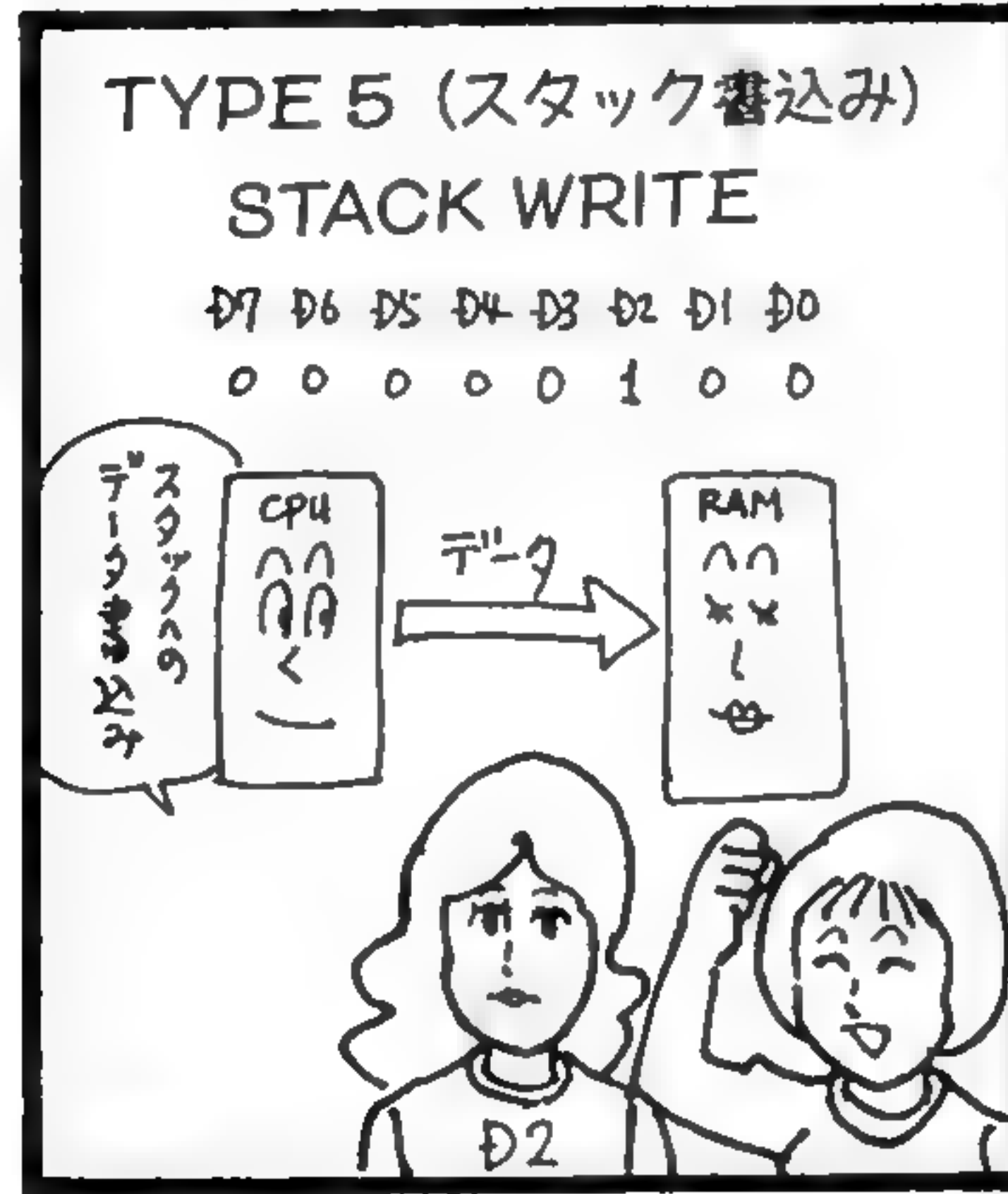
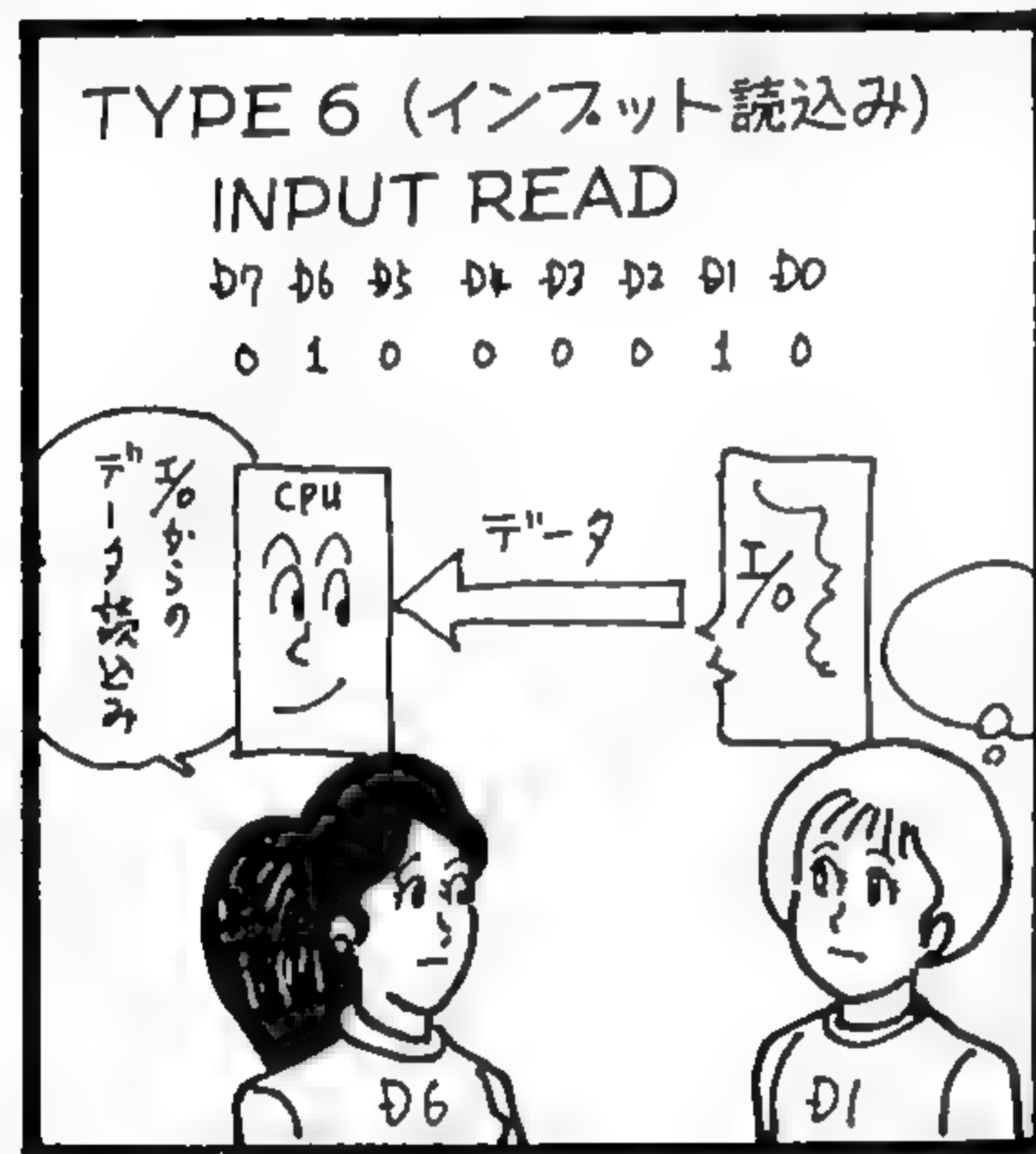
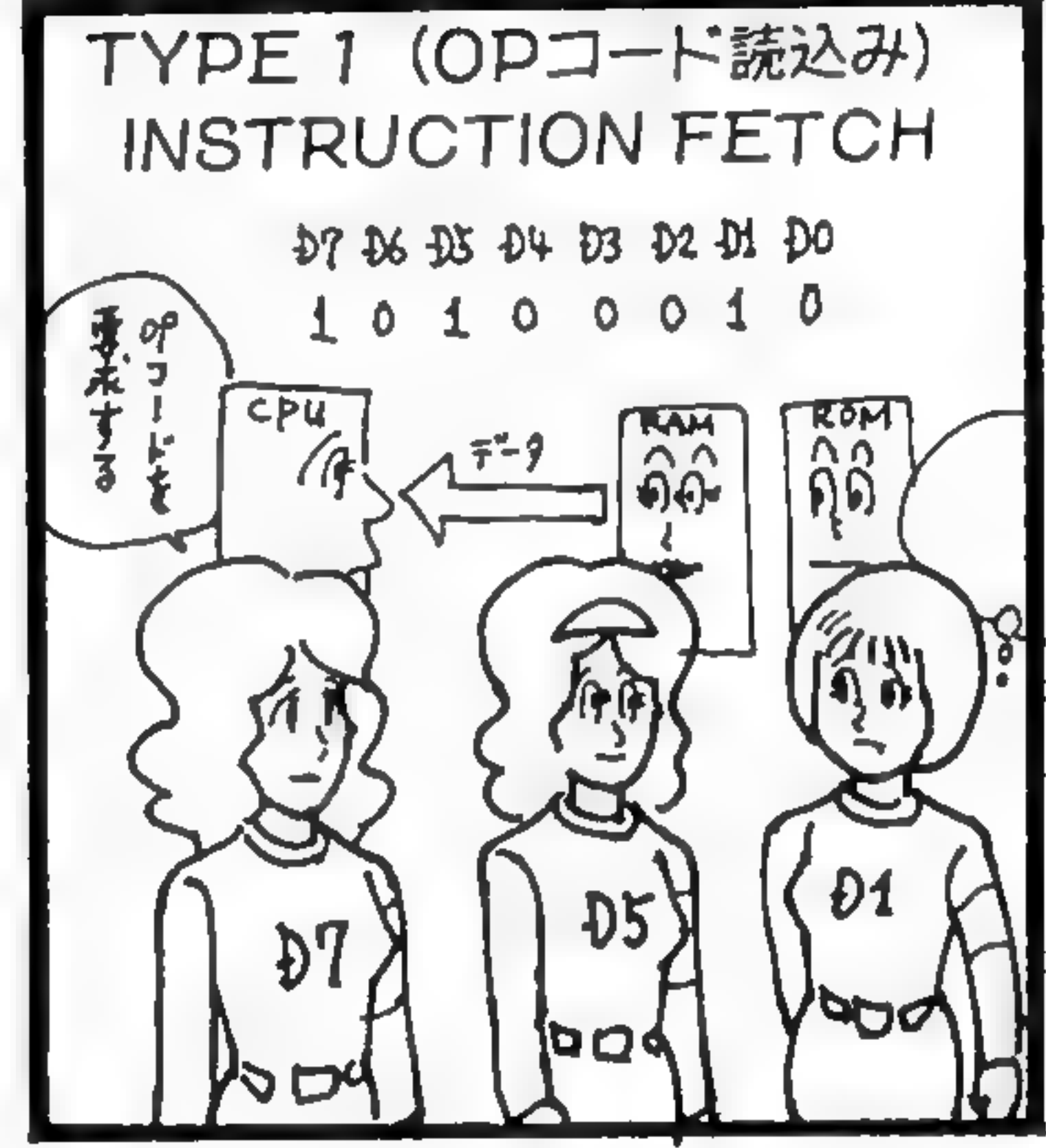
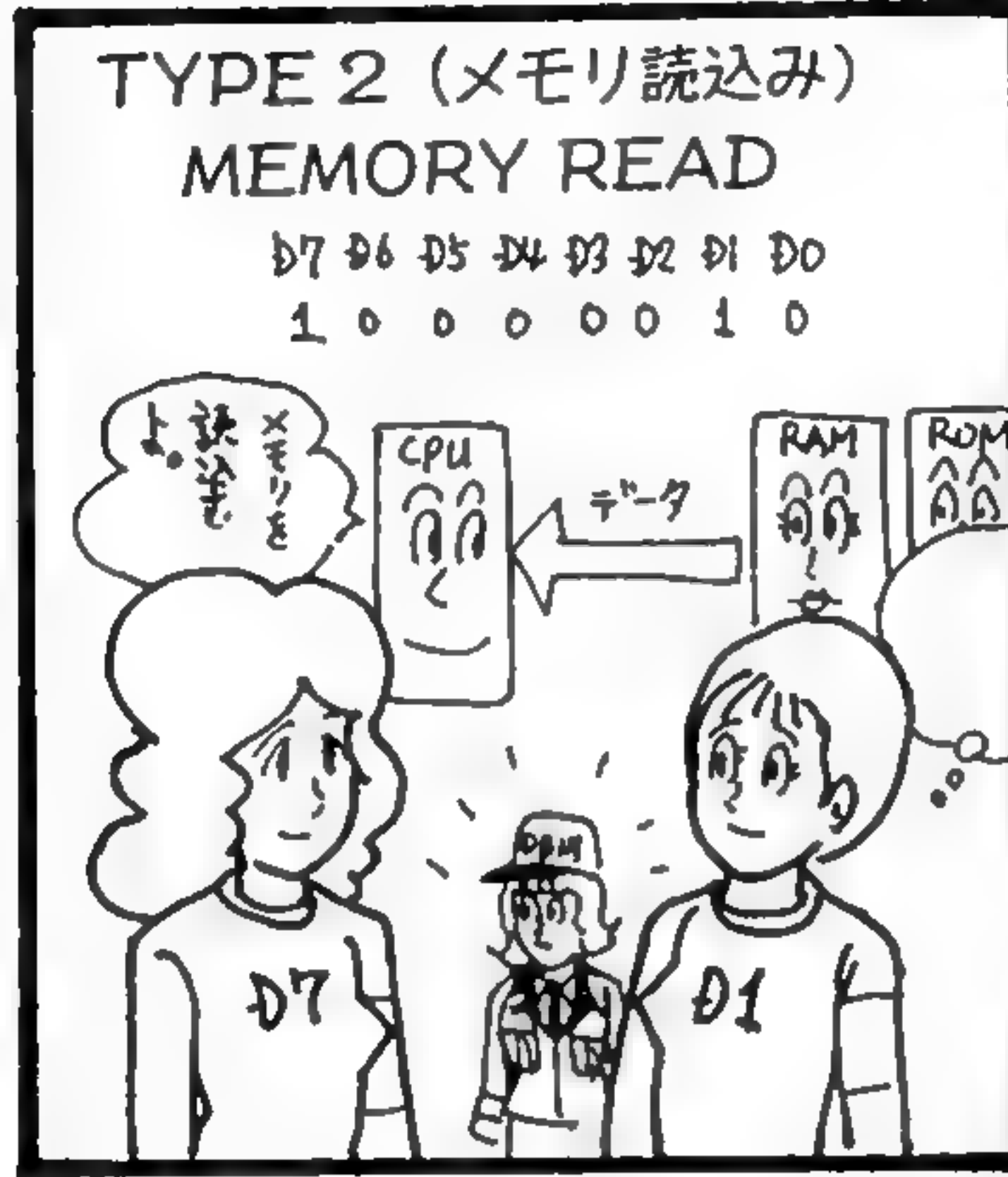
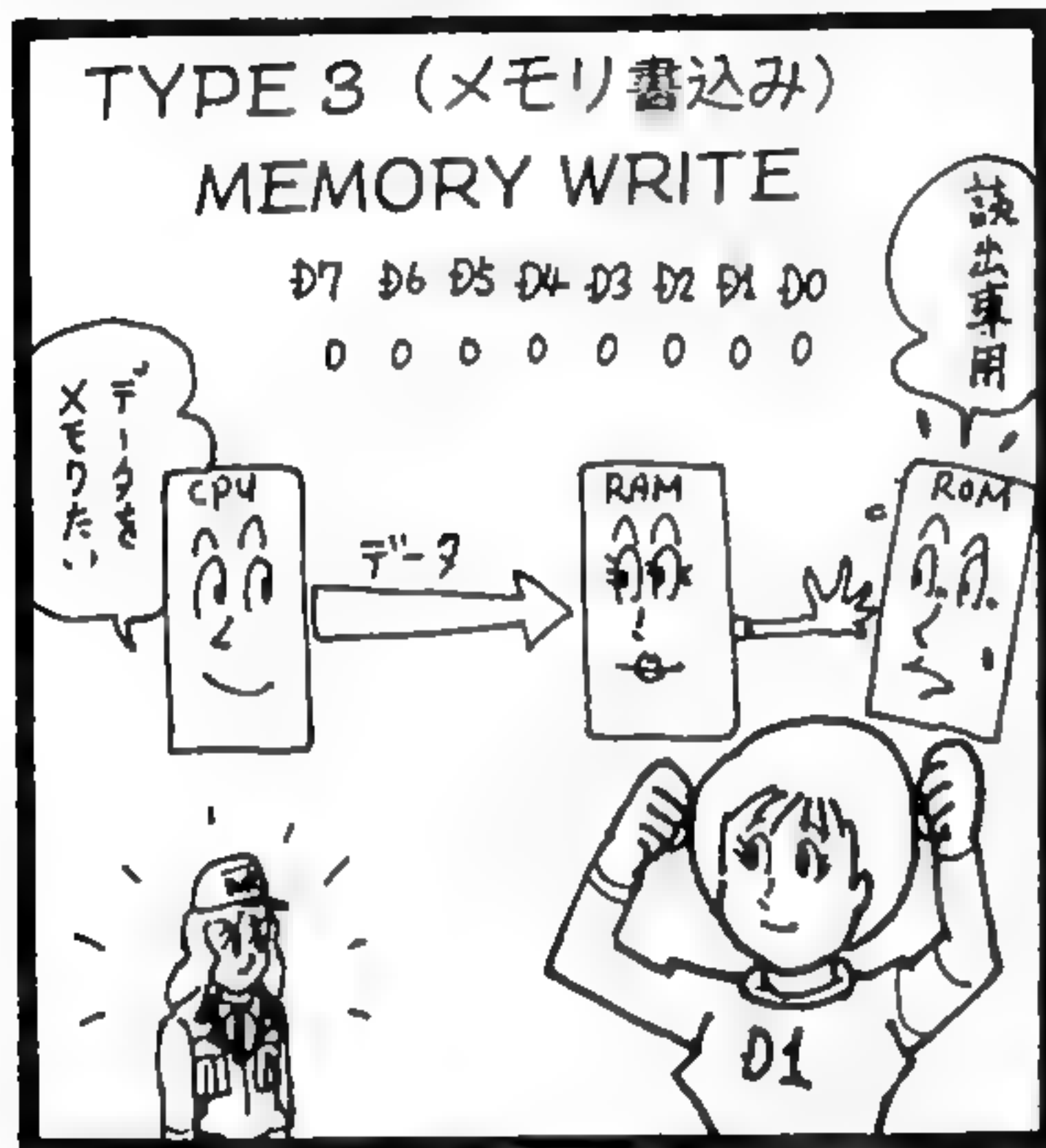
私は、このSYNCパルスをマシンサイクルごとに送っています。では友子ちゃんにボタンタッチしましょう。













次に、このステータスを使ってコントロール信号をつくり、データを外部メモリに書き込んだり、内部レジスタに読み込んだりしなくてはなりません。

ステータス → コントロール

ひとつの命令を実行するためにいくつかのマシンサイクルが繰り返されるわけですが、CPUからは常にこの10種の中のひとつのS・Iが最初に出力されているわけです。

TYPE 10 (HALT中の割り込み応答)  
INTERRUPT ACKNOWLEDGE WHILE HALT

D7 D6 D5 D4 D3 D2 D1 D0  
0 0 1 0 1 0 1 1

起る、た、ムニヤ

わかる、データ

割り込み

INT 00

長さは普通SYNCと同じ、φ2のクロック1周期分ですが...

タイミングとしてはSYNCちゃんのすぐあとに出ます。

SYNCの次は、私DBINです。私は、双方向性であるデータ・バスが、どちら向きになっているのかを知らせるんです。

データのバスは入力

DBINは、T2~T3のスタートの間出力されるということです。

データ読込みのタイミング  
CPUが勝手に読込む

マシンサイクルは、さらにクロック単位でT1、T2、T3といったステートに分けられます。

READYがくると、φ2の整数倍のクロック分だけ長くなり、足ぶみ状態になります。

新しい待てないよ

!!

READY

データ

これはアクセスの時間の長いメモリなどの、応答待ち用として考えられた機能なのです。

足ぶみ

CPUがデータを待つ

もし、READY信号がT2ステートでLOWになったら、何もしないTWというステートにし、CPUを足ぶみさせ、T3ステートに移らないようにします。

READY

私はDBINの信号を出すだけでなく、T2のステートでREADY信号がきていないかも見ているわけです。

T1 T2 TW T3 T4

READY



メモリへのデータ書込みはWRパルス全体を使います。TTLメモリのラッチよりも時間がかかります。スタティックRAMで250n(ナノ)sec以上のパルスを入れる必要があります。

データバス  
WR

この時間内にRAMのデータセット  
書き込250nsec以上

私はふつうT3ステートの間WRを出力するんです。

M1 M2  
T1 T2 T3 T1

phi1  
phi2  
SYNC  
WR  
データ  
アドレス

アドレス 時間

私はライト・メモリへの書き込みタイミングです。DBIN おねえさんよりタイミングは少し遅くなっています。DBINと同じマシンサイクルの中でいっしょに出ることはありません。

8080  
データバスの方向  
OUTモード  
WR

TK-80では、WAITとREADY端子を直結してONE WAIT動作をさせています。

WAIT  
READY

READYがきていると、TWステートに入り、T3ステートに移りません。これは最初ハード設計の時クロック周期や、メモリ速度をもとに何回待つが決められます。

T2 TW TW T3  
WR  
READY

私もT2ステートではREADYがくるのか見えています。

T1 T2 TW T3  
READY

このWR、DBIN、ステータスを組み合わせて、コントロール信号をつくっています。INTA、OUT、INP、MR、MW、MW(SP)はWR、DBINのどちらかに同期したパルスになります。

室温16℃きょうも雨が降ります。

クロックを1μsecにすると、TK-80のクロックより2倍もありますから、マシンサイクルごとの待ちステートは必要ありません。

10p 14 15  
X-TAL 9MHZ  
8224  
8080

TK-80キットの水晶発信器の周波数は18.432MHz、8224でも分周してますから、0.4882812μsecです。マシンサイクルごとにONE WAIT動作します。

0.4882812 × 2 μsec  
T1 T2 TW T3 T1  
WR  
データ

マシンサイクルM2に移りましょう。

これは、MOV A, M ね。

DBINでHizを解除OUTモードに3

こうして、あおして、こうなって……説明はいりませんね。

ROMちゃんは、自分ではなにもできないのねえ。

私は+1インクリメント

こんなマンガありますか？

フーんだ

マシン・サイクル M1  
MOV A, M

phi1  
phi2  
SYNC  
STSTB  
データバス  
ステータスメモリ  
アドレスバス  
DBIN

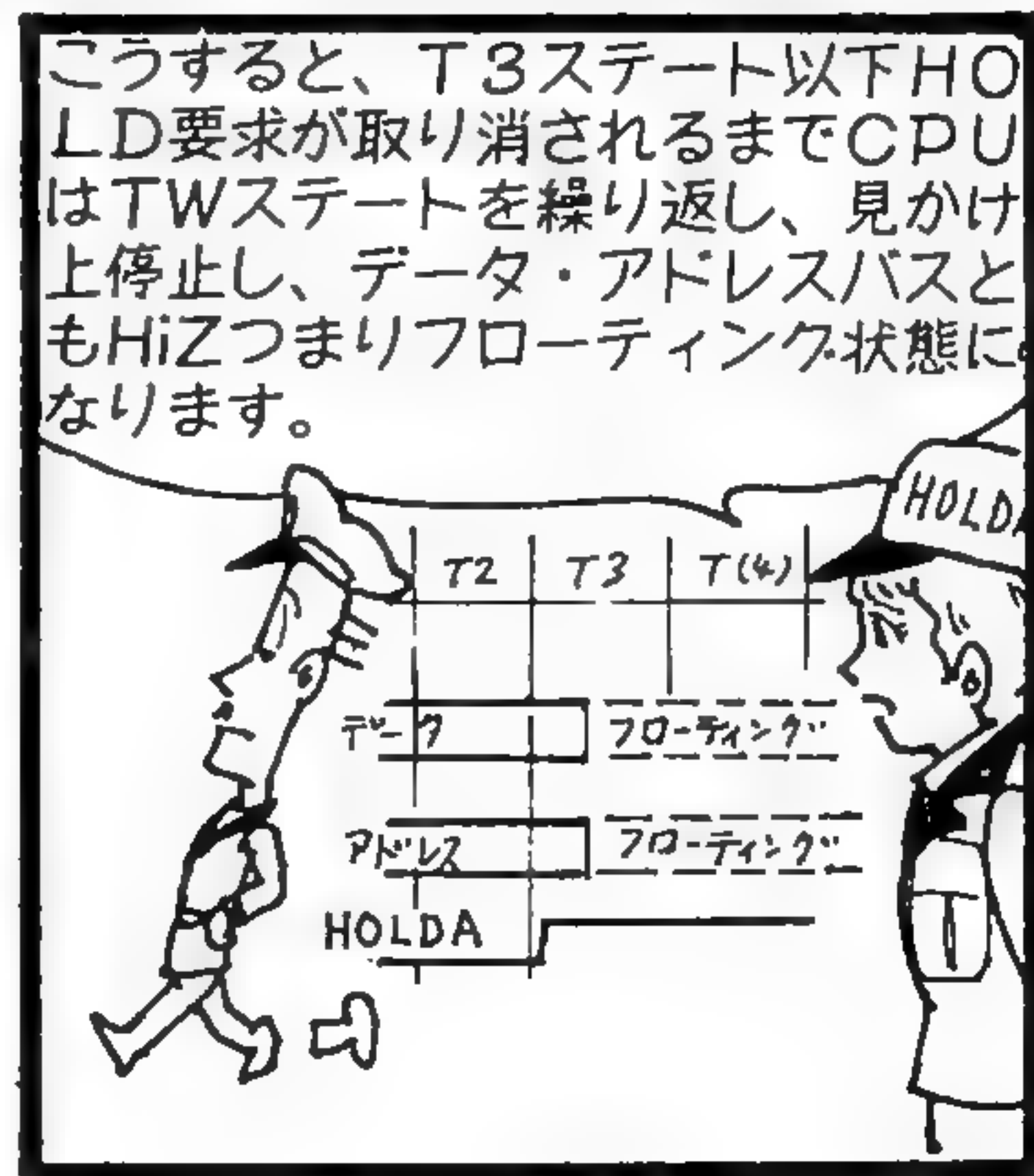
7E  
7E  
(PC) = 2073

CPUが読む









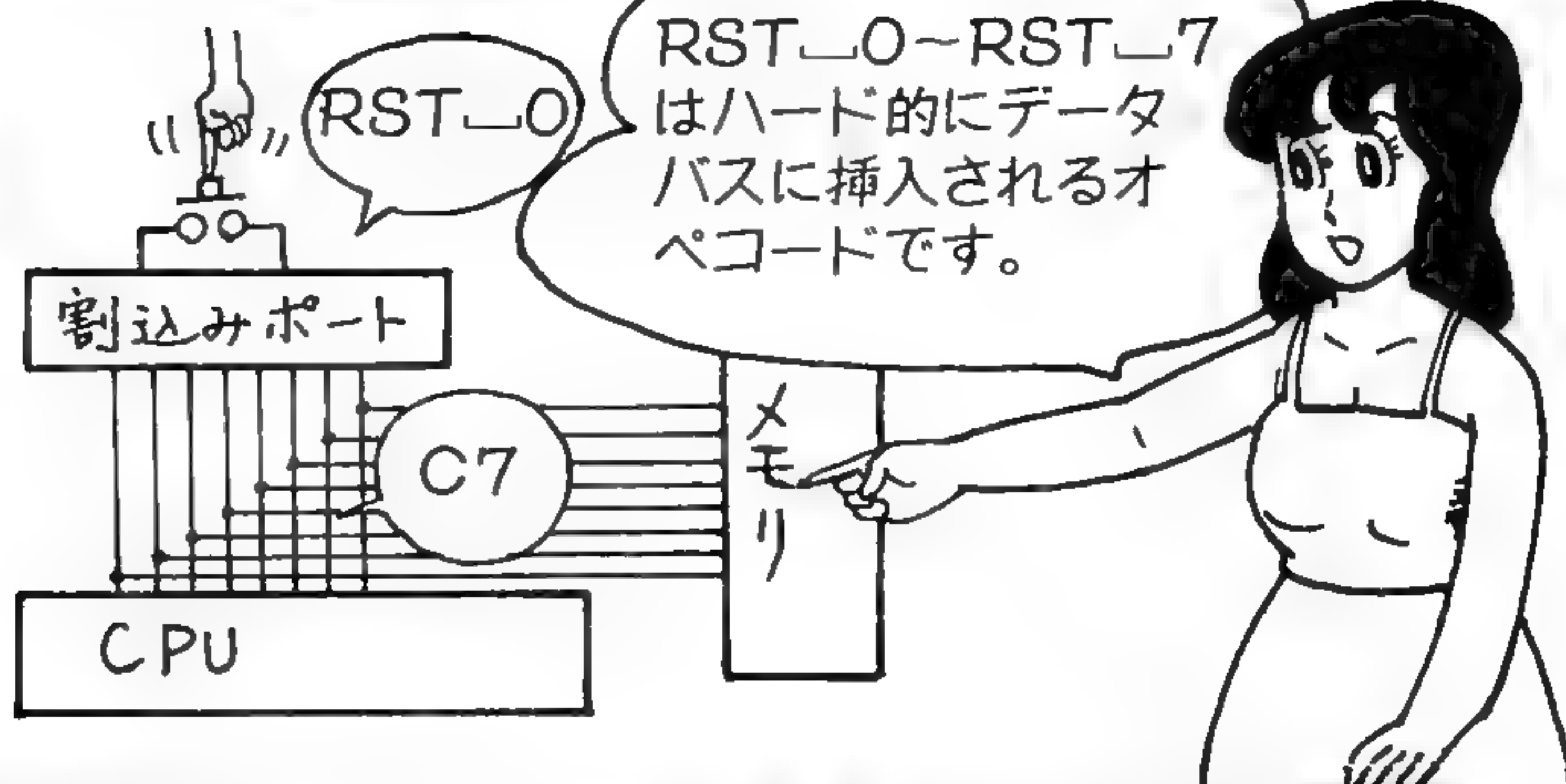




## ⑩ハード的なオペコード

インタラプト

### (割込命令)





割り込み許可を与えるのはソフト、つまりプログラムです。EI(Enable Interrupts)という命令を使います。マシン語は11111011B (2進表現) F B H (16進表現)

INTEはインタラプト・イネーブルの略です。割り込みが可能かどうか外部に知らせます。

8080には、INTEとINTの2つの割り込みのためのピンがあります。

ハード的にコントロールできる8つのサブルーチンという感じです。

必要ならばEI命令を入れて戻す。  
必要ならばEI命令を入れて戻す。  
必要ならばEI命令を入れて戻す。  
E1命令を入れておく。  
E1命令を入れて戻らないとこれ以後の割り込みはできない。  
独立した4つのプログラムです。

この図は、CPUがある時間ひとつのプログラムしかやってないことを示しています。プログラムカウンタがひとつしかないので当りまえですね。

8080の場合割り込みプログラムは8つまでとることができます。

私も働くの。  
プログラムカウンタは私だけなのよ。もち、割り込みプログラムも。  
スタックポイント

そして、そのプログラムの実行が終わるとリターン命令によって再びメインプログラムに帰るというものです。一度割り込み状態になるとINTE状態が取り消されるので割り込みプログラムの終りにEI命令を入れておくことが必要です。

この図はプログラムの長さもなく実行時間も短く示しています。

たとえば割り込み3というボタンを押してやるとメインプログラムの実行が、一時中断されて、割り込み3で指定したプログラムが実行されるのです。

サブルーチンと呼び出すのはCALL命令でしたね。

割り込み(サブルーチン)プログラムの場合、命令コードを1バイトにするために8つの先頭番地は固定されているんです。

(H) 番地 メモリ(単位はバイト)  
0000 ←0  
0008 ←1  
0010 ←2

CALL命令コード (3バイト命令)  
↓  
CD B7 02

CALLされるサブルーチンの先頭番地、(02B7番地) ここにはOPコードが入っていません。

サブルーチンの場合、そのプログラムの先頭番地はCALL命令の中に含まれていましたね。

OPコードがある番地  
201A CD B7 02  
201B CD  
201C 02  
201D  
201E

サブルーチンプログラム  
02B7  
02B8



ね。

割込み レベル	2進表現の opコード	16進の opコード
0	11 000 111	(C7)
1	11 001 111	(CF)
2	11 010 111	(D7)
3	11 011 111	(DF)
4	11 100 111	(E7)
5	11 101 111	(EF)
6	11 110 111	(F7)
7	11 111 111	(FF)

NNN

RSTn命令の一般的な表現は  
11NNN111です。  
このNNNに0~7の3ビット  
を入れると、それぞれのオ  
ペレーションコードができま  
す。やってみまし  
よう。

RSTn

このnは0~7までの  
数字。たとえば  
RST5

0、1、2、3、4、5、6、7の  
8つの割込み番地は、3ビット  
でRST命令に挿入され、  
決定されます。

000	...	0
001	...	1
010	...	2
011	...	3
100	...	4
101	...	5
110	...	6
111	...	7

そして、そのあと、プログラムカ  
ウンタの内容を  
0000 0000 00NN N000  
とするんです。このNNNは0~  
7を表わす3ビットです。たと  
えばNNN=3とすれば、  
00000000000011000

0 0 1 8  
16進表現

0018番地が割込み  
3の最初のOPコ  
ードの場所になります。

CPUはこのRSTn命令をフ  
ェッチすると、プログラムカ  
ウンタの内容をHとLの2バ  
イトに分けて、おのおのスタ  
ックポインタをさしている場  
所に移します。

PC RAM Xメモリ

H L

L H ← SP

この8つの1バイト命令は、今  
まで説明したプログラムメモ  
リの中にはありません。別の所  
からデータバスにのせて  
CPUにフェッチ  
させ  
ます。

RSTnの  
opコード

C7, CF, D7, DF  
E7, EF, F7, FF

つまり0000~003Fまでは、  
普通のプログラムはメモリで  
きません。また8バイトメモリ  
では割込み自身のプログラムも  
できないのでジャンプ命令で他の  
場所にとばします。

Xメモリ

0018 C3  
38  
30

JMP 3038

そして、とばした  
先に割込み3のフ  
ログラムを書いて  
おけばいいのです。  
プログラムの終り  
にはリターン命  
令をつけてPCを元  
に戻します。

3038

C9 ← RET  
リターン命令

Xメモリ  
戻る

またよそ  
見して、  
しょうの  
ない子。

電源  
リセット  
時もこ  
からスタート

0

割込み3  
このopコードは  
DF (H)  
この結果  
プログラム  
カウンタは  
0018となり  
ます。ここ  
最初のopコ  
ードを入れます。

1 2 3 4 5 6 7

0000  
0008  
0010  
0018  
0020  
0028  
0030  
0038  
003F  
40FE  
40FF

データバス

8バイトおきに  
8レベル、合計  
64バイトがフ  
ログラムメモ  
リの  
0000番地か  
ら003F番地  
にあることにな  
ります。

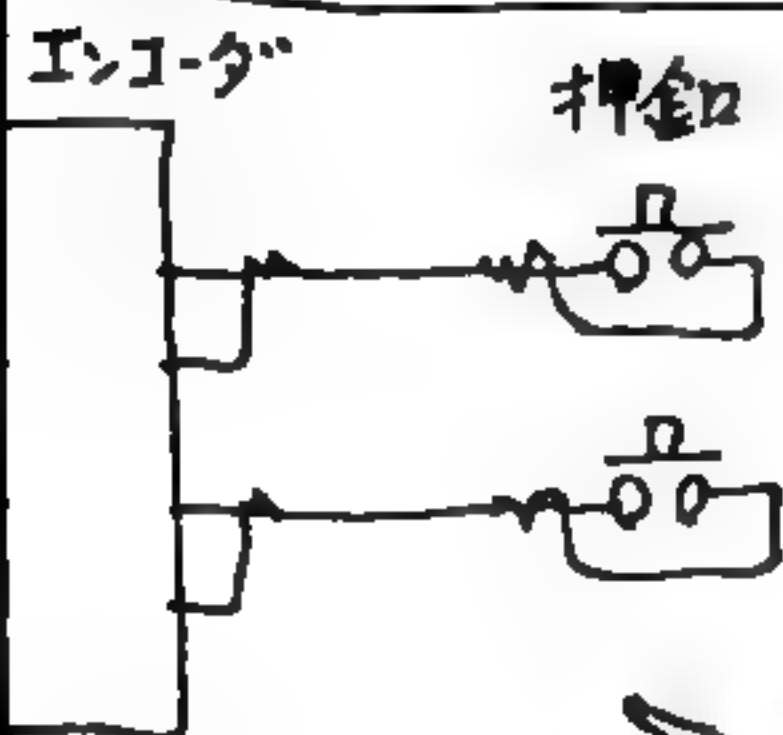
0018  
3 F F A  
H L

4 0 F E  
L H

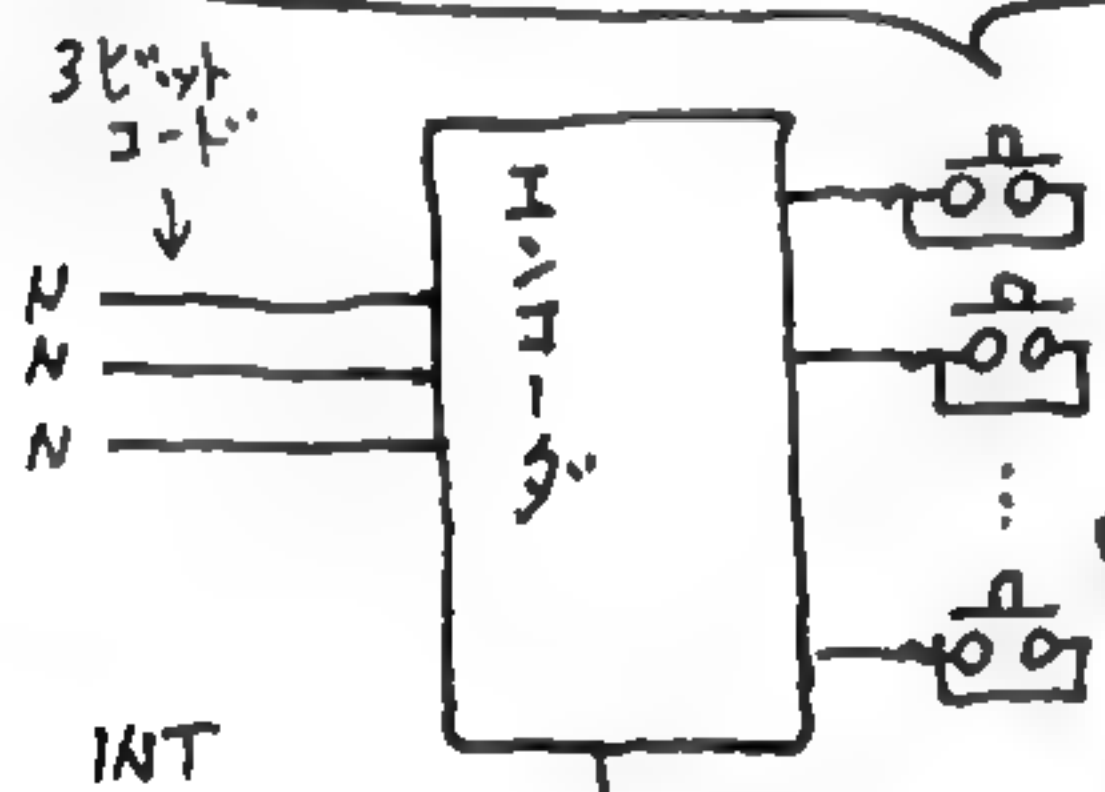
SP



割込みの信号は、ハードから任意の時刻にやってくるんです。それはリレーの接点とか、押釦の接点とかその他いろいろな機械的なもの、電子的なものです。



各接点信号は優先順位をつけるエンコーダによって3ビットコードに変換されます。ここでNNNをつくるわけです。同時にCPUへのINT信号を出します。



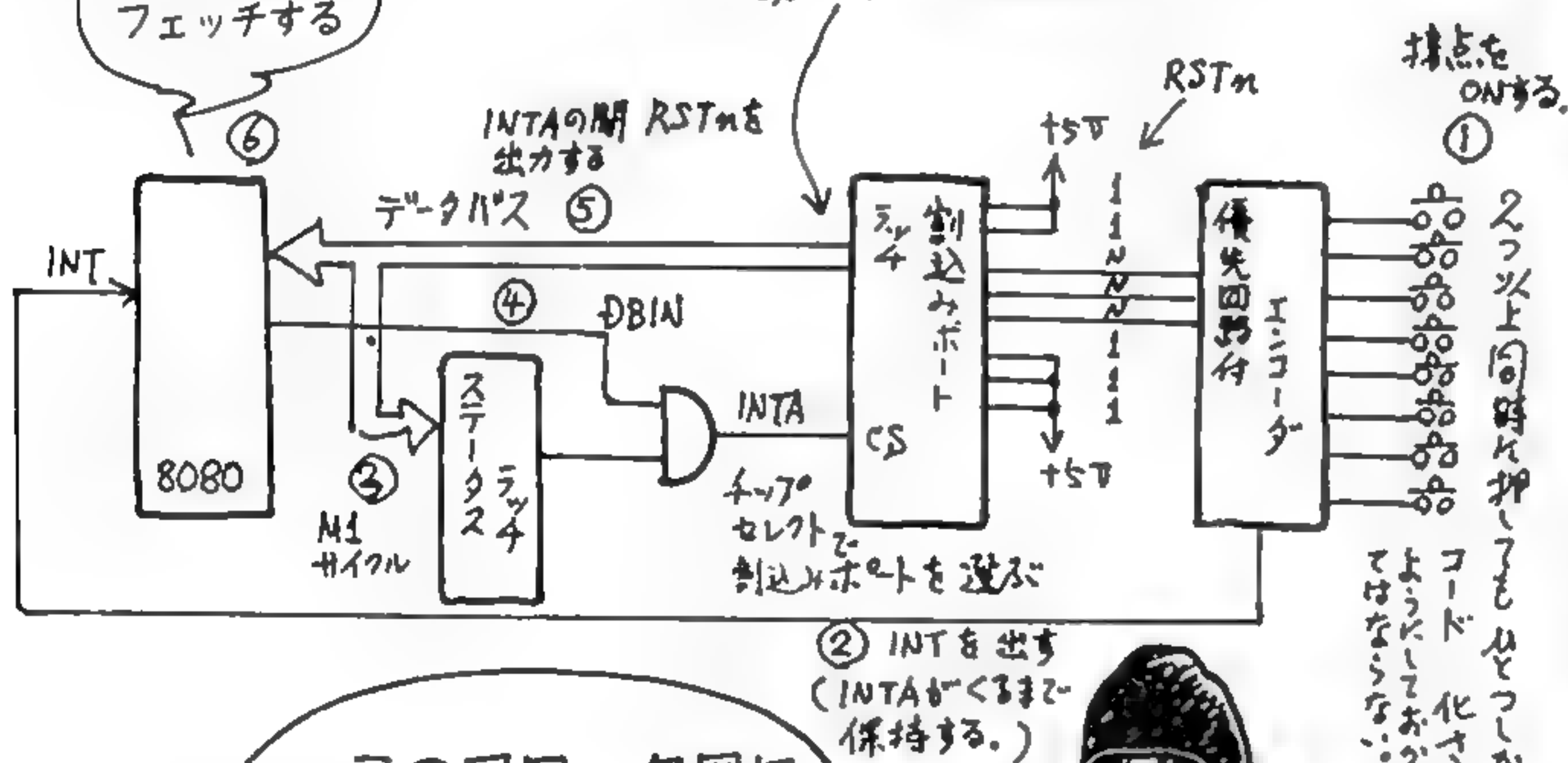
押釦を2つ以上に同時に押ししてもどれかひとつしか選ばれないようになっている。

では、次にRSTn命令をCPUにフェッチさせる方法についてお話ししましょう。



RSTn命令をフェッチする

ここはトライステート・ゲートであるCSに入るとHが解除される。

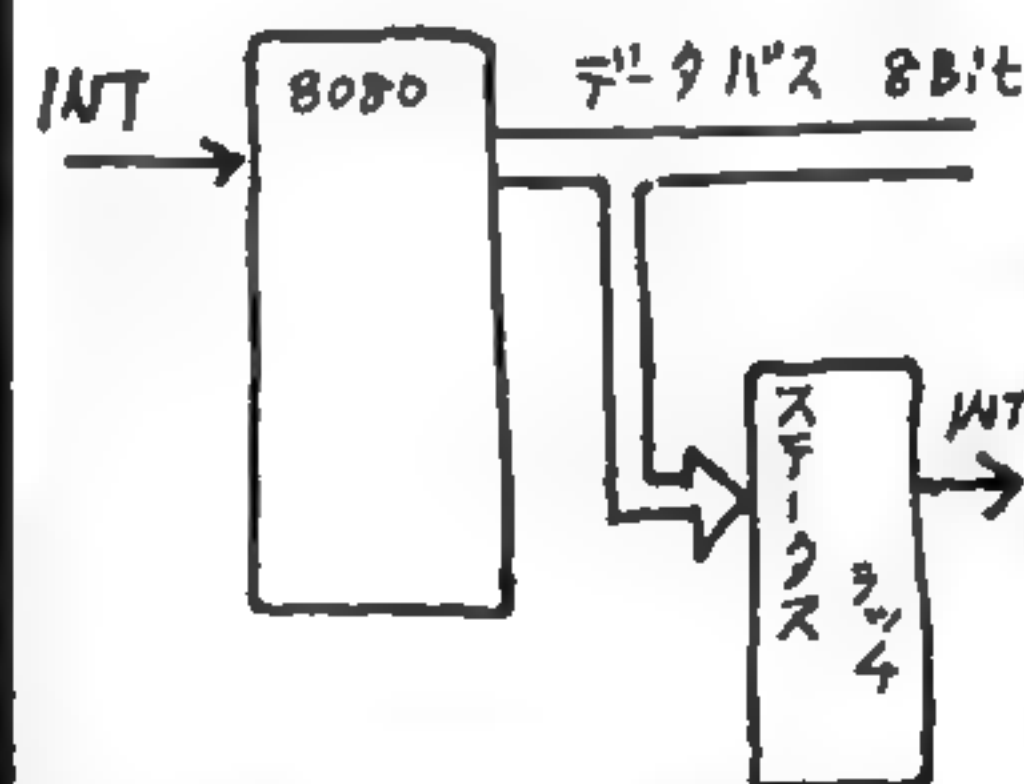


このブロック図に示された番号の意味を次のコマのタイミングチャートで説明しましょう。

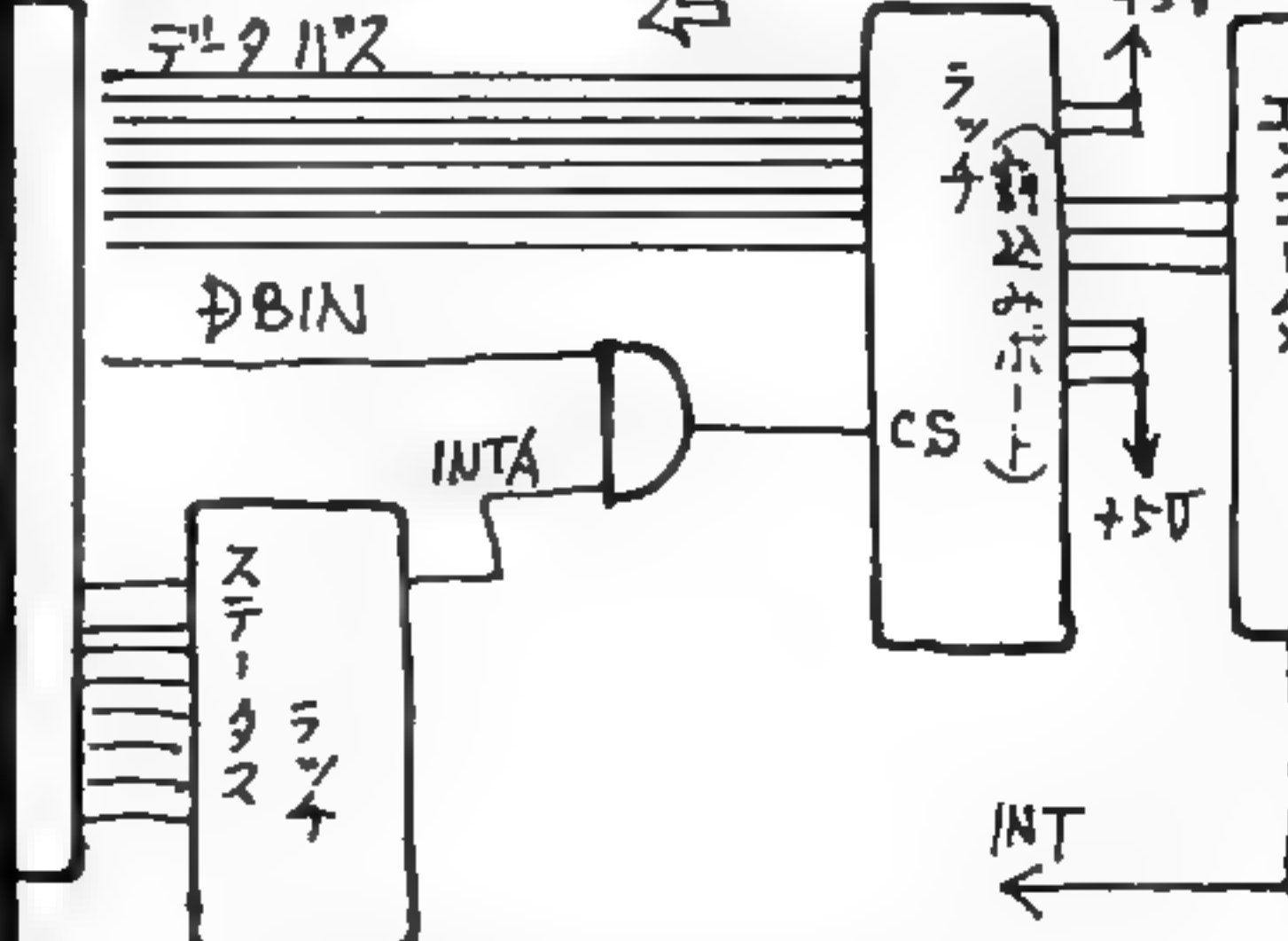
ハードを読むためには時間と信号の必要があります。



このINTがCPUに入力されCPUがINTE状態にあると次のマシンサイクルM1 (OPコードフェッチサイクル) のステータスとしてINTAがデータバスに出力されます。



このステータスのINTAと、CPUがデータを読み込むタイミングの出力DBINとのANDをとって、データバスにRSTn命令をのせるのです。









ただ、ここで私たちはポートという新しい言葉に出会いましたね。私たちはこれからCPU←メモリから、さらに外の世界を広げていくのです。

8080の周辺デバイス(部品)として、フロッパブル割込みコントローラ8259というLSIがありますがここでは触れません。

8080 周辺デバイス

8259

これでだいたい割込みについての基本的な説明は終わりです。ここではRSTn命令は、外部から割込みポートを通してデータバスに入ってくると覚えてください。

これは、データバスがポートを通して外部と連絡しています。

これが、いままでの世界、CPUがやっていることは私たちにはわかりません。(目で確認できない)。

チップセレクト  
トリステスト・ゲートをあける。メモリはこの状態から解放する。

これぐらいで8080の割込みについての説明は終わりたいと思います。なおZ80などではもっと多彩な割込みができますので関心のある方は勉強してください。

この2つをまとめて書くとこうなります。データバスがようやくバス(共用車)らしくなってきたようです。

コントロールWAとINTAによって、それぞれのトリステストゲートを開ける時刻が違っているのでバスが混乱することはありません。



## ⑪入出力装置用ポート (I/Oポート)

マイコンイコールパソコンと思い込んでいる人も多いのではないのでしょうか。マイコンはこのようなかたちで工場の中でも活用されているのです。

マイクロコンピュータはソフトとハードを切り離して説明できないところがあります。IN命令、OUT命令もハードを抜きにしては説明しにくいのです。

I/Oポートへの8ビットデータの入出力は僕Aレジスタが窓口になります。

メインメモリ  
65Kバイト

8080やZ80には0000~FFFF(65Kバイト)の他に00~FFまでのI/O用ポートを指定できます。

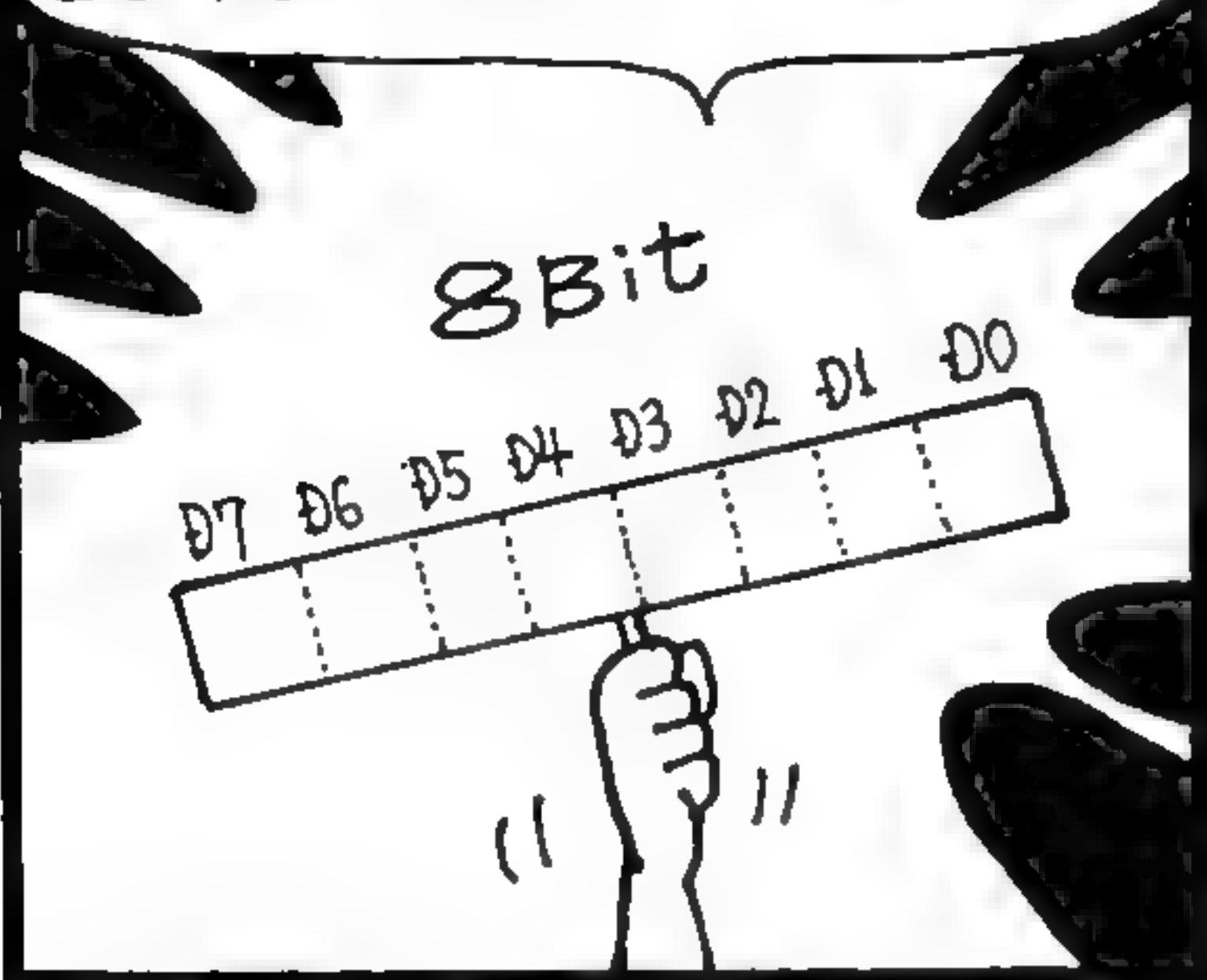
256  
バイト

256  
バイト

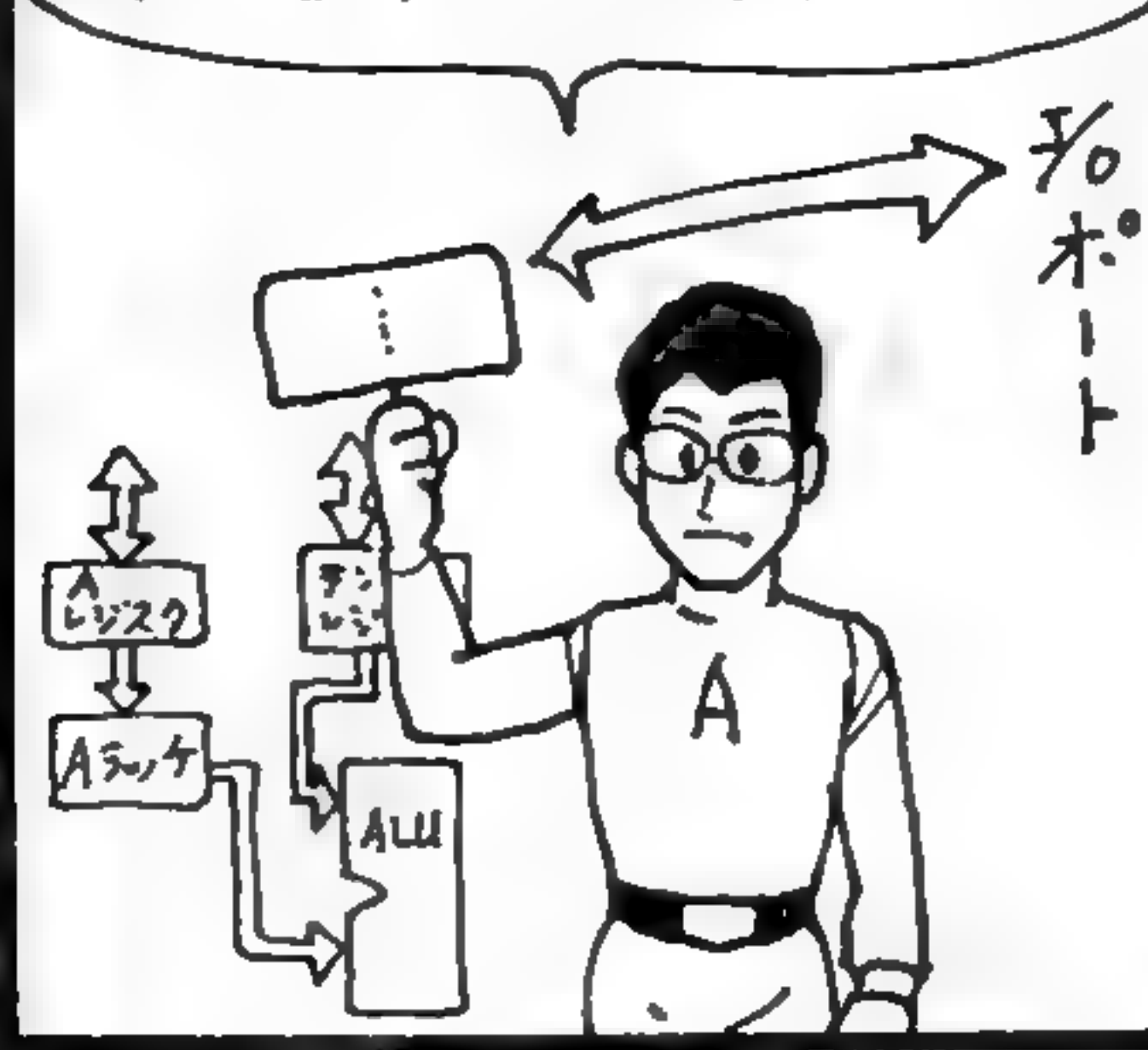




僕が一度に扱えるデータは8ビットです。I/O命令の時は各ビットに特別な意味を持たせます。



8080のINPUT(入力)命令と、OUTPUT(出力)命令の2つは、僕と外部デバイスとのやりとりなのです。



こんにちは。僕はAレジスタです。女ばっかしのこのマンガの中で貴重な存在。

8080の内部のお話

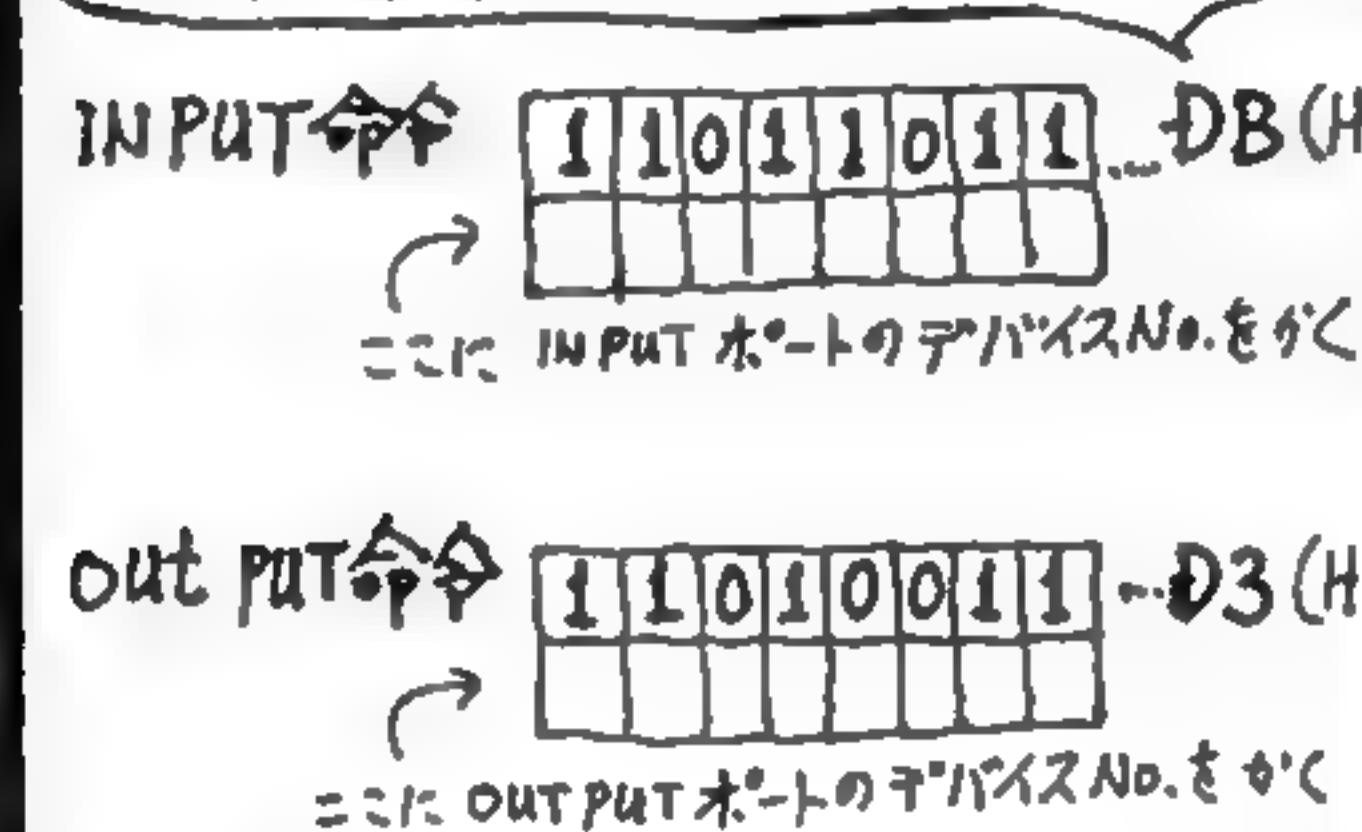


1バイト(8Bit)では0~255(D)の256種のポートを指定できることになります。

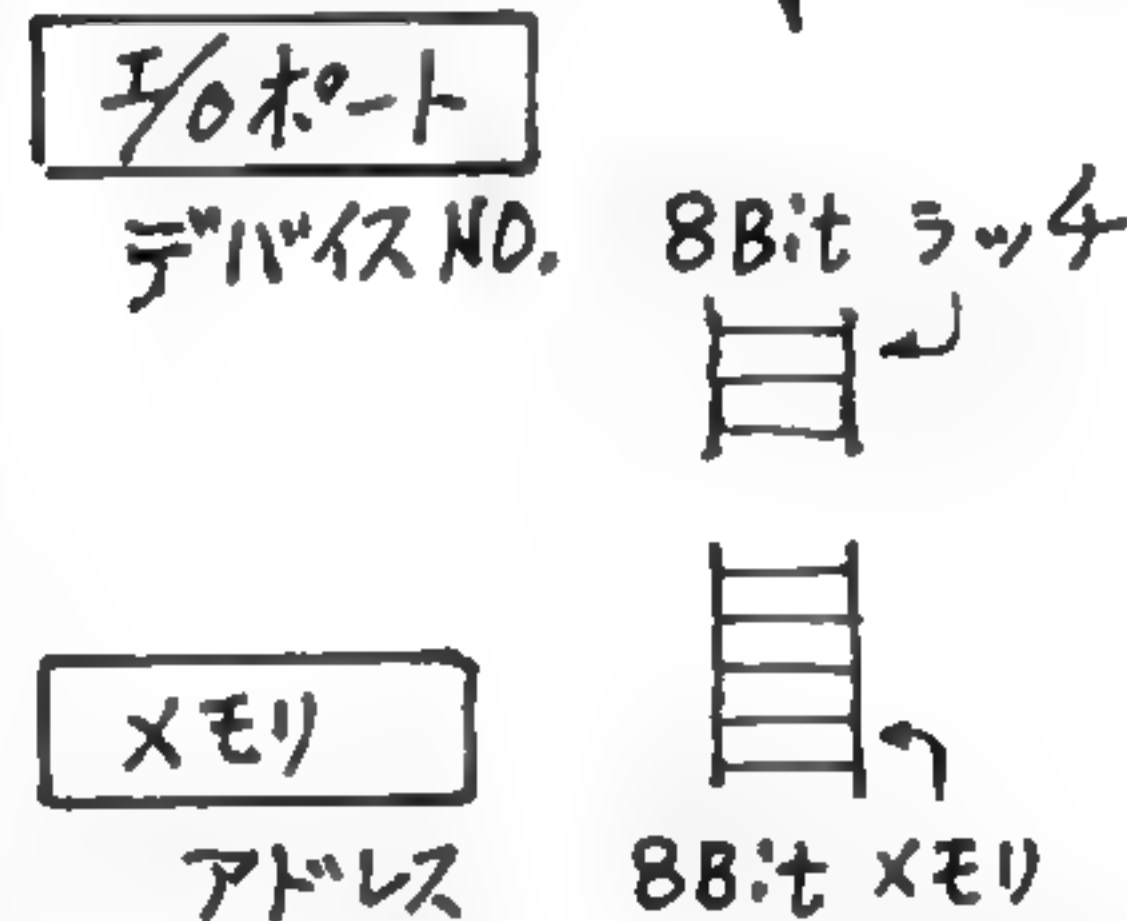
256 { 0000 0000 ... 00 (H)  
1111 1111 ... FF (H)



メモリの番地に相当するポートのデバイスNo.は命令の2バイト目に書いておきます。IN、OUTの2つの命令は2バイト命令です。



IN、OUT 2つのポートを構成している部品はデバイスといい、メモリの番地と区別するためデバイスNo.と言ったりします。



I/OにもI/O表、つまり、デバイスNo.に割り当てた信号名を記入したものがが必要です。

メモリにはその内容を示すアドレスマップが必要でしたね。

Aレジスタから出力  
3内容

OUT PUT表 (INPUT表)

DIV No.	D7	D6	D5	D4	D3	D2	D1	D0
00								
01								
02								
03								

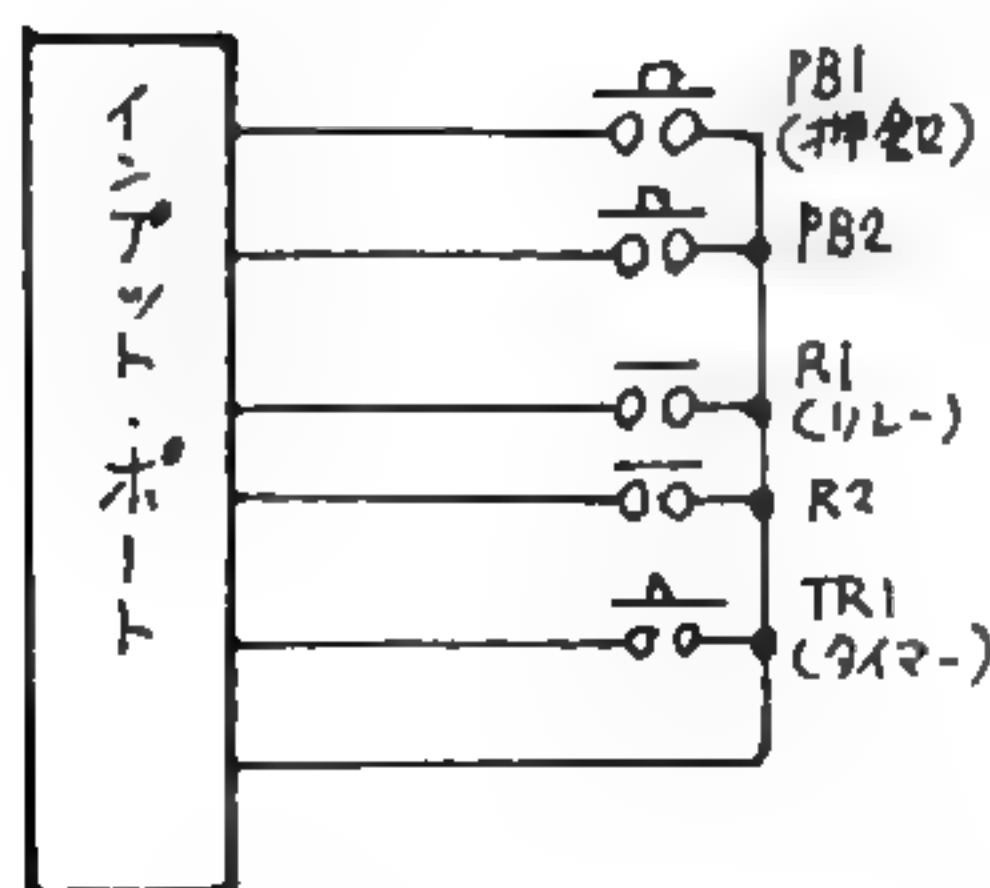
命令の、2バイト目で指定する。

でも、ふつうの制御にマイコンを使用する場合、そんなにたくさんのINPUTやOUTPUTが必要なわけではありません。だってボタンなら2,048個分なんですもの。

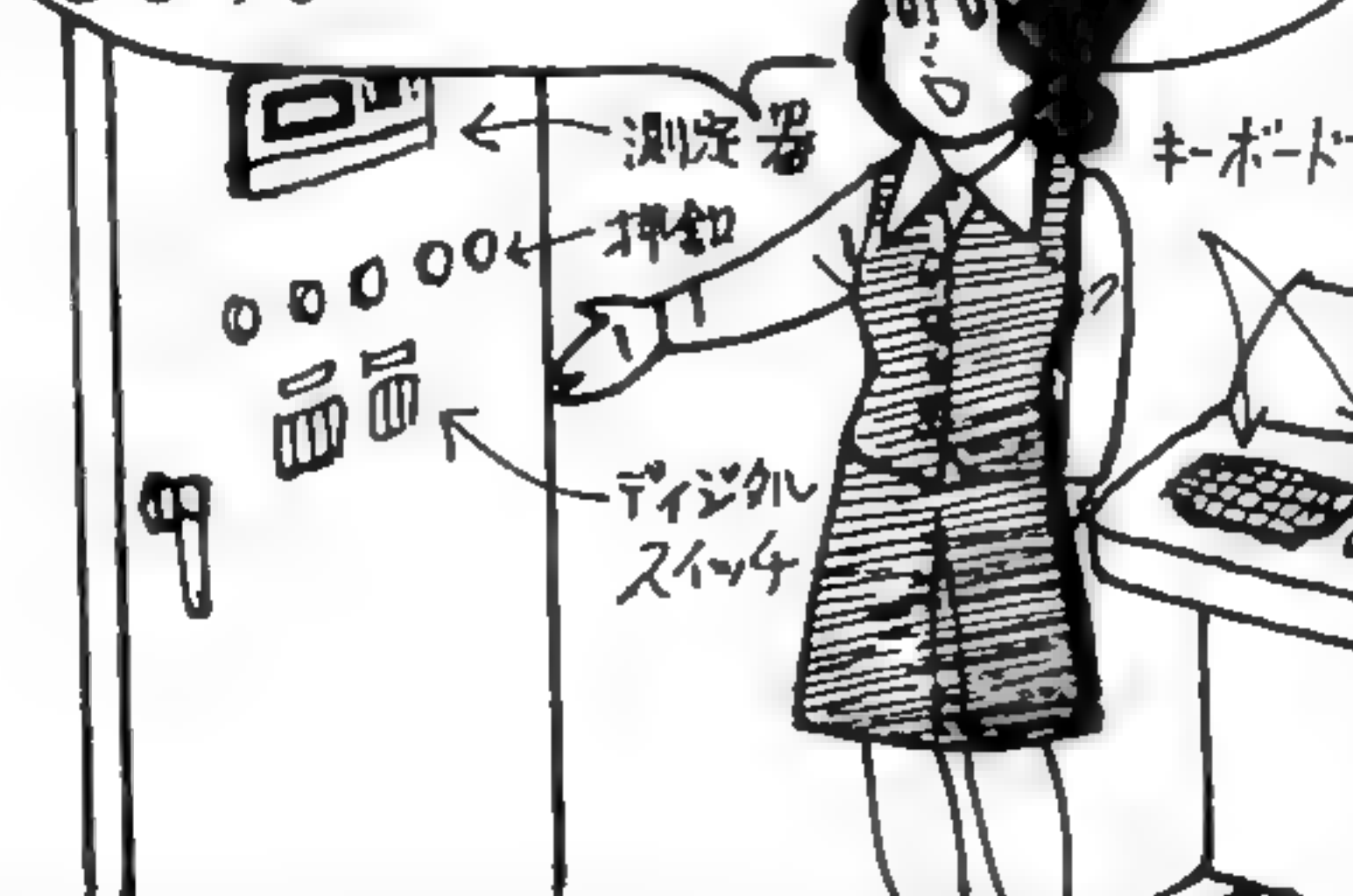
$$256 \times 8\text{Bit} = 2,048$$



外部入力



マイコンへのインプットとしては制御のためのボタンとかリレー接点とかリミットスイッチとがあります。また計測データなどはBCDコードでインプットします。



では、まずインプット命令からお話しましょう。ここでは INPUT・インプット・入力はそれぞれ同じ意味で使っていますから固く考えないでください。その時の気分で使い分けているだけです。





D7にフリントの押ボタンを割り当て  
そのデバイスNo.を  
〇〇としました。

たとえば、D0~D3に  
ディジスイッチのデー  
タを、

INPUT表

DIV No	D7	D6	D5	D4	D3	D2	D1	D0
00	フリント 押ボタン				ディジ スイッチ	8	4	2 1
01								

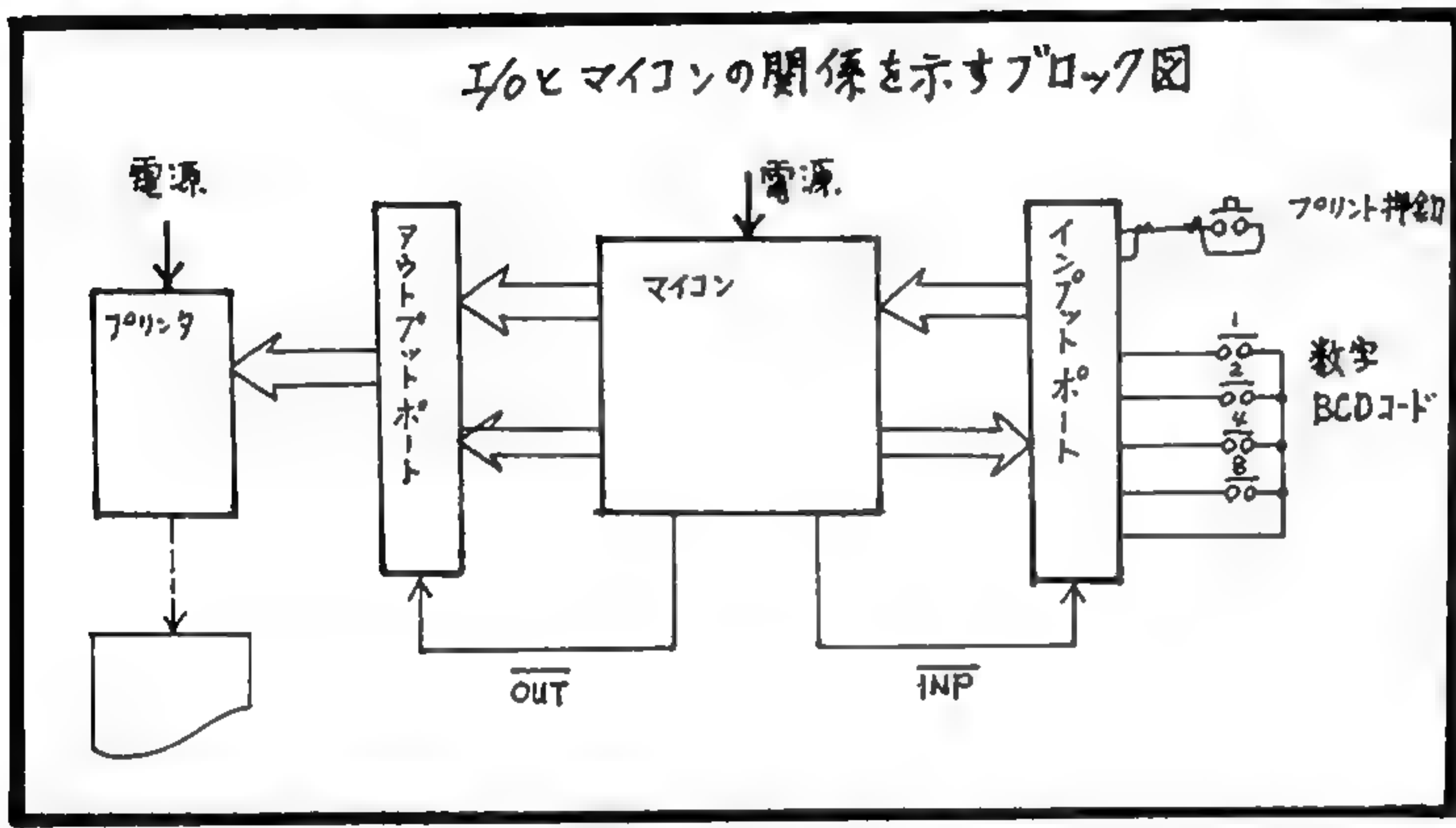
作る製品の仕様が決まったら、  
入力するものも決まりますか  
らINPUT表をつくります。

インポート機器

フリント押ボタン

ディジスイッチ  
1桁(BCD)

A 8  
B 4  
C 2  
D 1  
COM



数字1桁をマイ  
コンを通してフ  
リントすること  
でI/Oを勉強し  
ましょう。

マイコン

フリント

ディジ  
スイッチ

フリント  
押ボタン

オープンコレクターの場合は、  
積分回路になるので応答速度が  
遅いんです。

データ  
Bit

5K

コントロール

遅れ

オープンコレクターのNAND  
ゲートでもいいんですけど...

コントロール

データ  
バス

D0  
D1  
D2  
D3  
D4  
D5  
D6  
D7

入力

インプットの場合、このポート  
はデータバスに出力する側にな  
ります。だから出力がHiZの状  
態のとれるICを使わなければ  
なりません。

BUS

コントロール・バスにinpとい  
う1 Bitの線(信号)が出ていま  
すね。このポートのセレクトには  
このINPを使いま  
す。

DBIN

INP

コントロール  
HOLD A

このコントロールは、やはりCPU  
からのステータスと読み込みタイミ  
ングのDBINを使います。

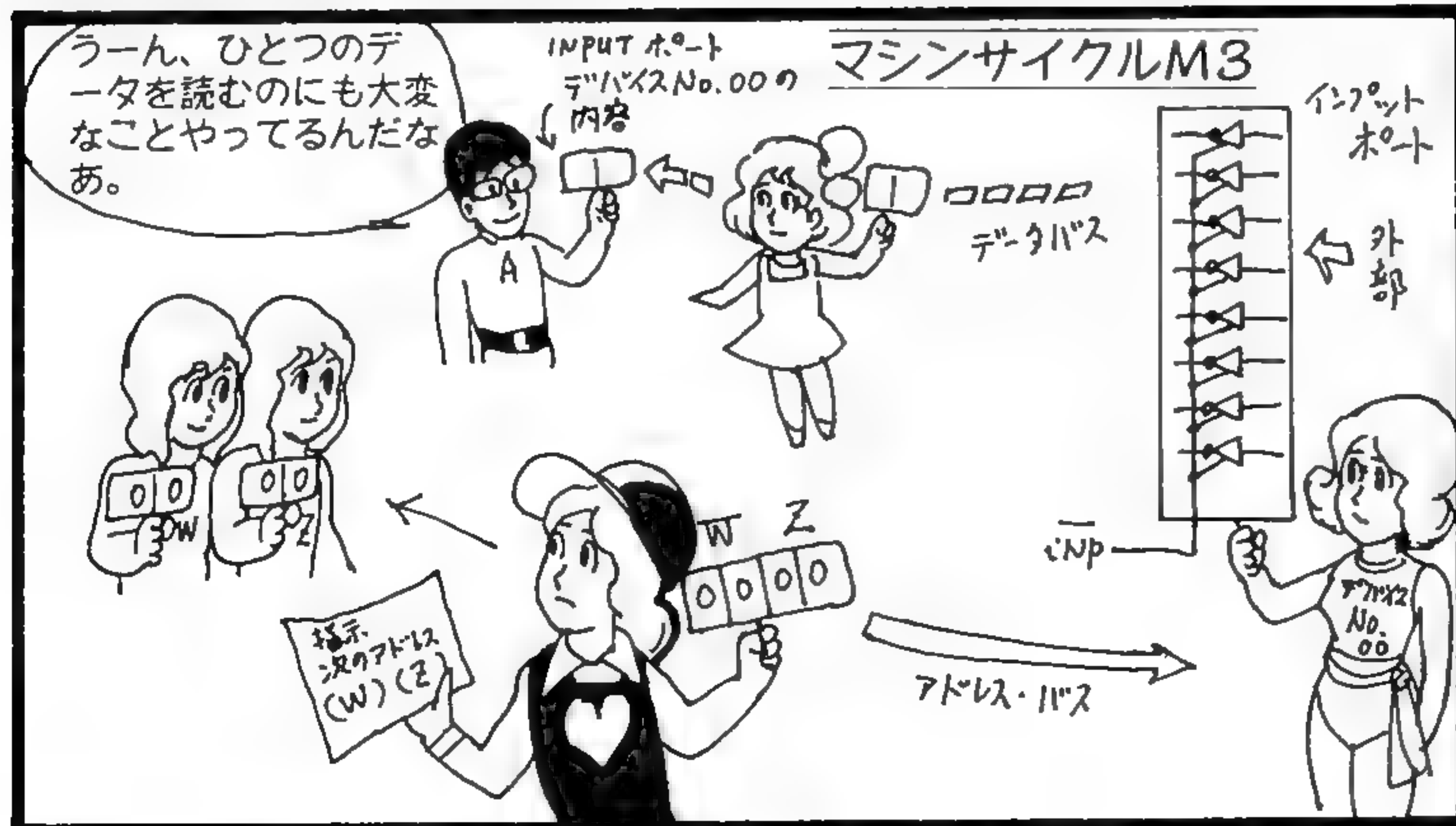
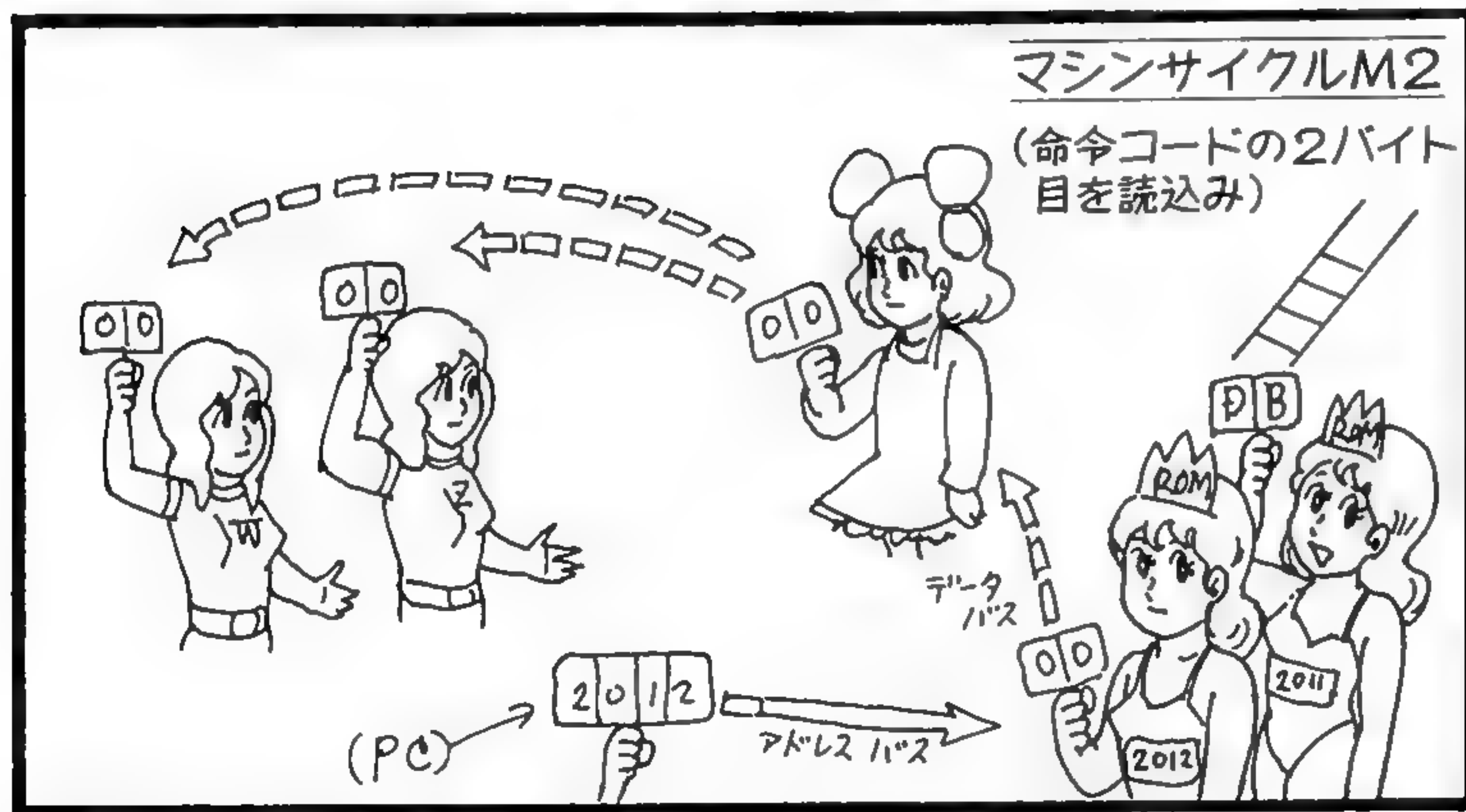
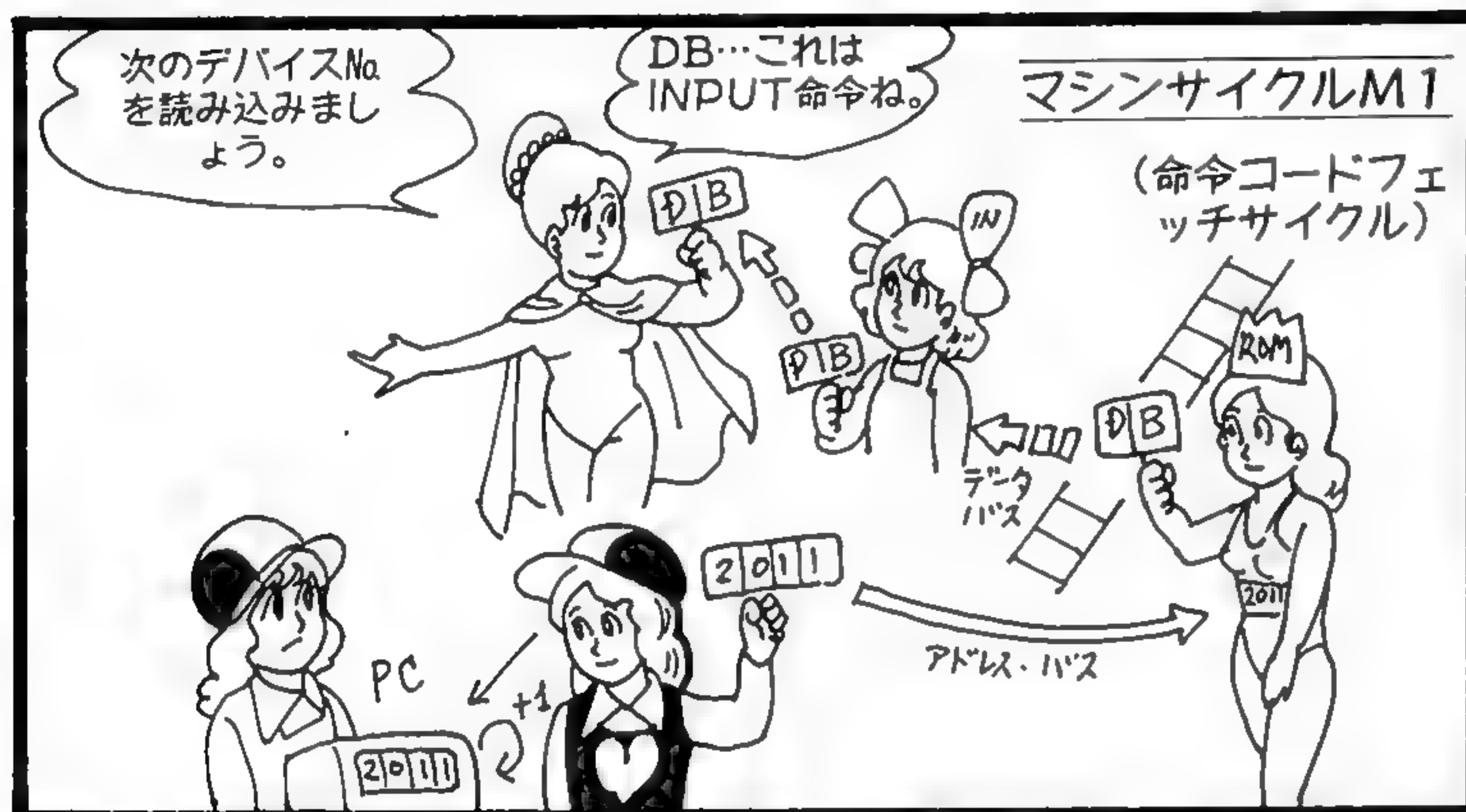
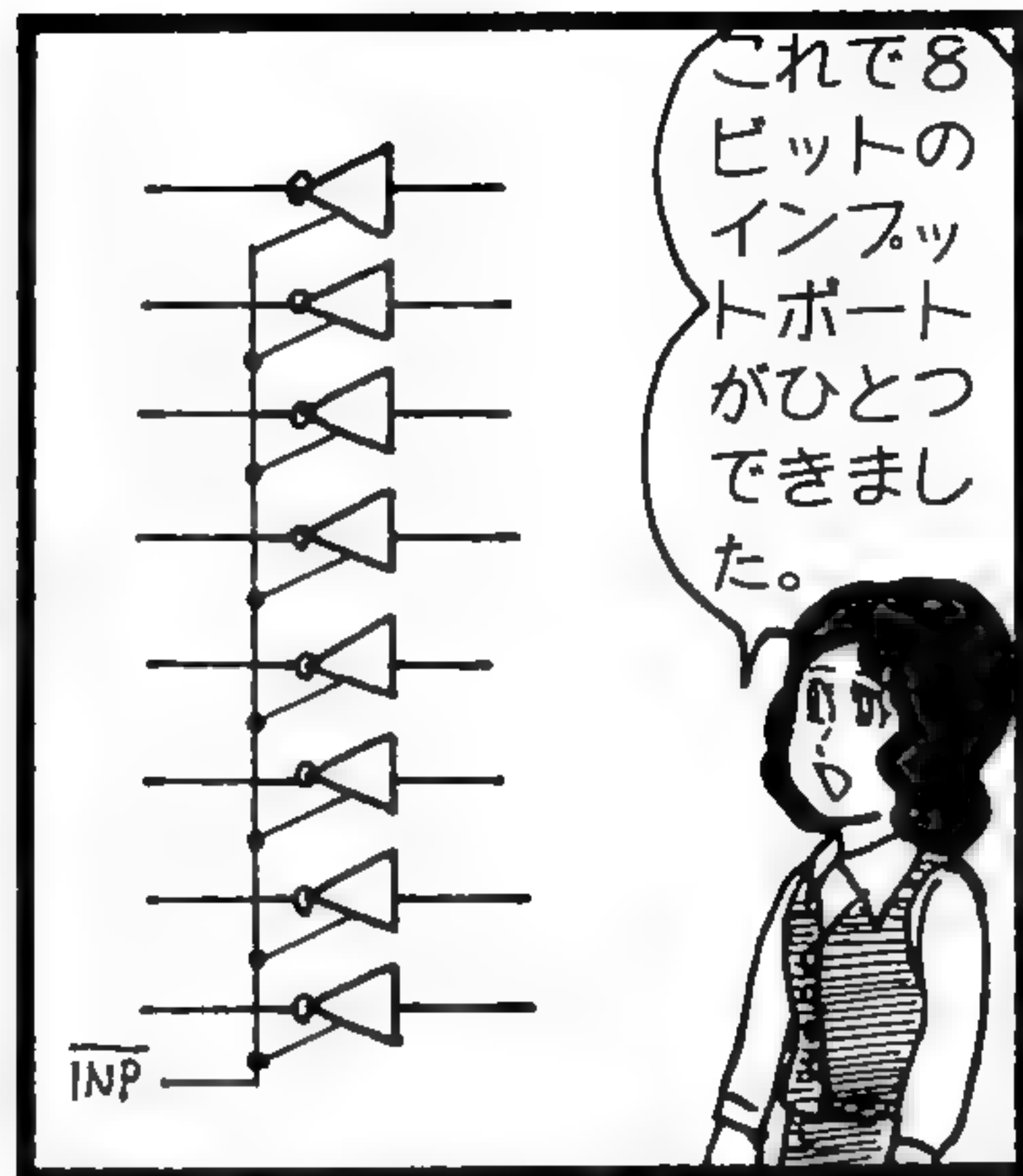
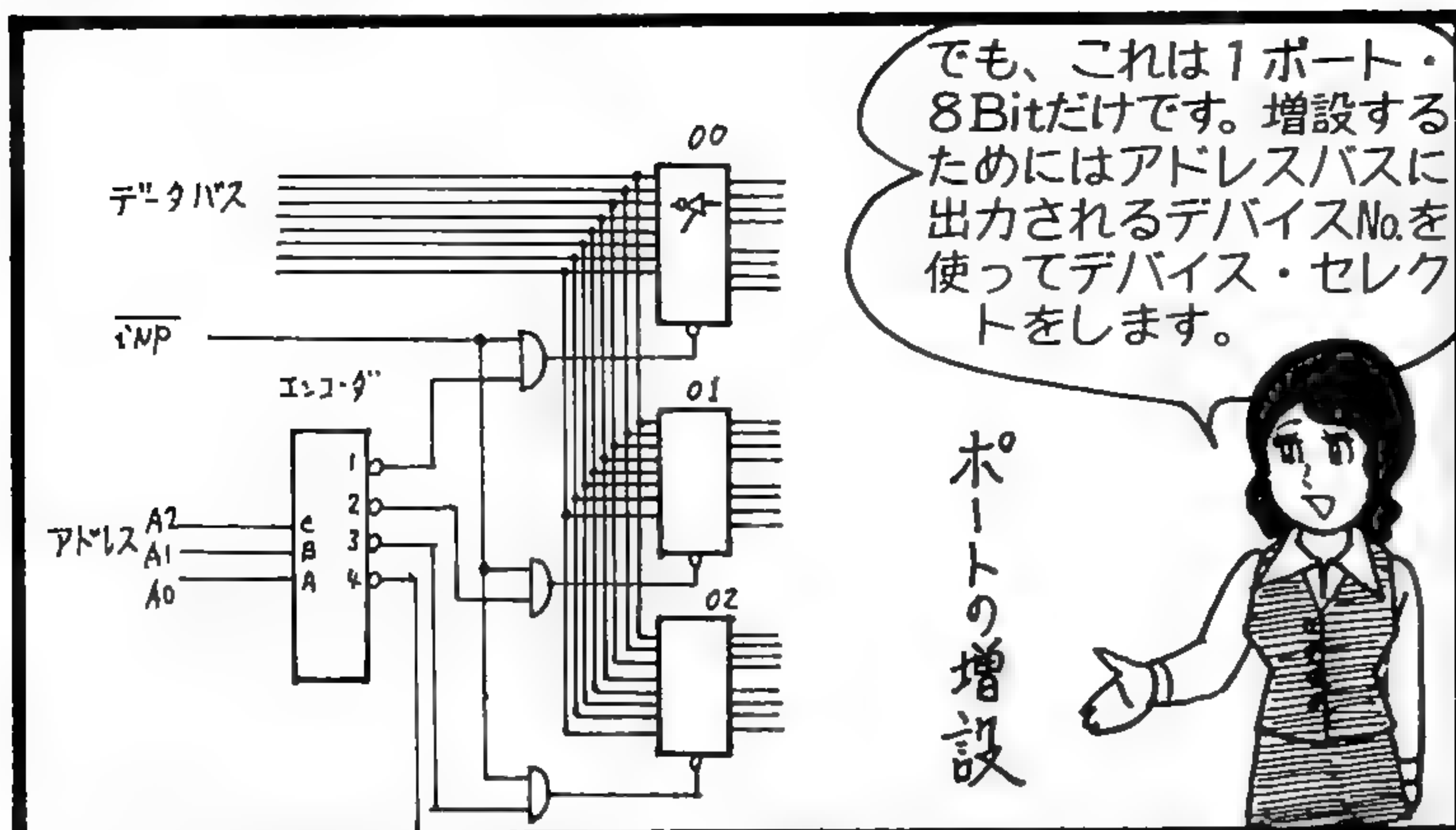
そこで、やっぱりトライステート・  
ゲート、これは、オープンコレク  
ターの欠点改良のために考えられ  
たものです。

D1  
D2  
D3  
D4  
D5  
D6  
D7  
D8

入力

コントロール







INPUTポートのデバイスNo.0  
0に割り当てたプリントボタン  
のビットはD7です。

DEV No.	D7	D6	D5	D4	D3
00	1	0	0	0	0

このビット  
を調べる

大ざっぱに書く  
とこんなフロー  
チャートになり  
ます。

```

graph TD
    Start(( )) --> Input[INPUT]
    Input --> Decision{?}
    Decision -- NO --> Input
    Decision -- YES --> Output[OUTPUT]
    
```

プリントボタンは  
押されたか?

ところで、プリントボタンを  
押したかどうかCPUがどう  
して判断するのでしょうか。  
これはソフトの問題です。

これは、2バイト命令で2バイト  
目のデータとAレジスタの各ビット  
ごとのANDをとるという  
命令です。

レジスタ  
E6  
80

結果を  
とります。

ゼロ  
フラグ

73ページの命令一覧表を見  
てください。この下の方にA  
NIというのがありますね。

E	6
1	1
1	0
0	1
1	1
0	1
1	0

実行  
(A) ← (A) ∧ (B2)  
(C) ← 0  
A ... 論理積

ここに「1」がたったかどうか  
見るためにはAレジスタに読  
み込んだデバイスNo.00のデ  
ータと、10000000、つまり80H  
とのANDをとってその結果が  
0かどうかの判断をすれば  
いいんです。

ソフトによる  
ANDの方法

つまりボタンが押されるまで、  
ゼロのフラグは上がりっぱなし  
なので、ここにジャンプ命令を  
入れて下げた時に通過するよう  
にすればいいのです。

ポートデバイスNo.00  
D7  
AND  
80  
プログラム

ZERO

そして、その結果が0か、  
そうでないかで、フラグが  
変化するのです。ここが、  
ソフトで判断できる  
ポイントなのです。

この命令は8つの  
ANDゲートをソフ  
トで実現するこ  
とができるのです。

まだ他にもAレジスタの特定  
のビットを調べる方法がありま  
すが、要は、そのビットを操作  
してフラグを働かせ、そのフラ  
グを条件にしたジャン  
プ命令を使うこと  
なんです。

これがCPUに制御信号をプロ  
グラムでフェッチさせる方法で  
す。割り込み命令と、どこが違  
うか自分で確かめてみて  
ください。

プログラムで  
ハードの変化を  
キャッチする

ANDをとって、条件ジャンプ  
させるわけですね。

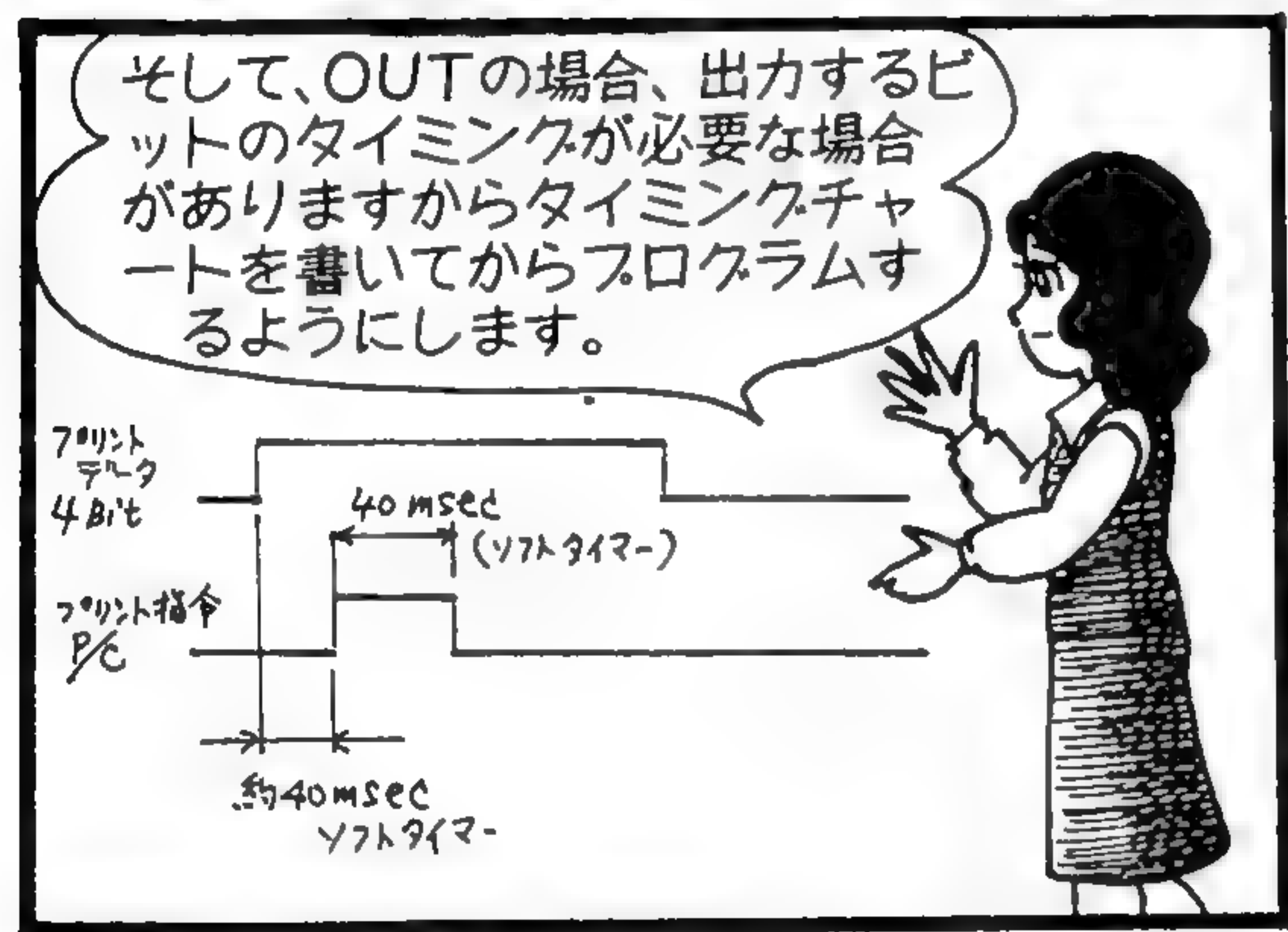
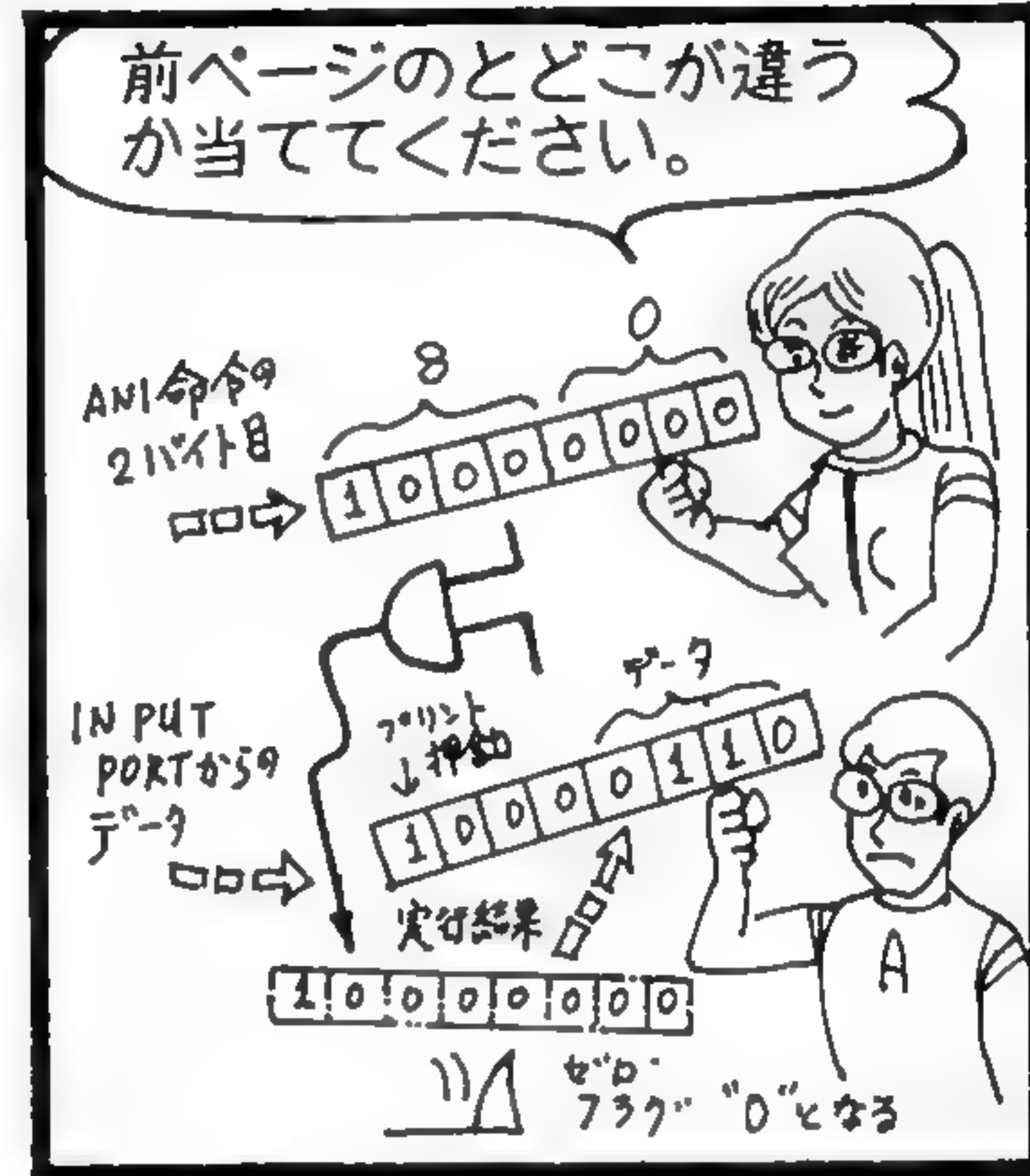
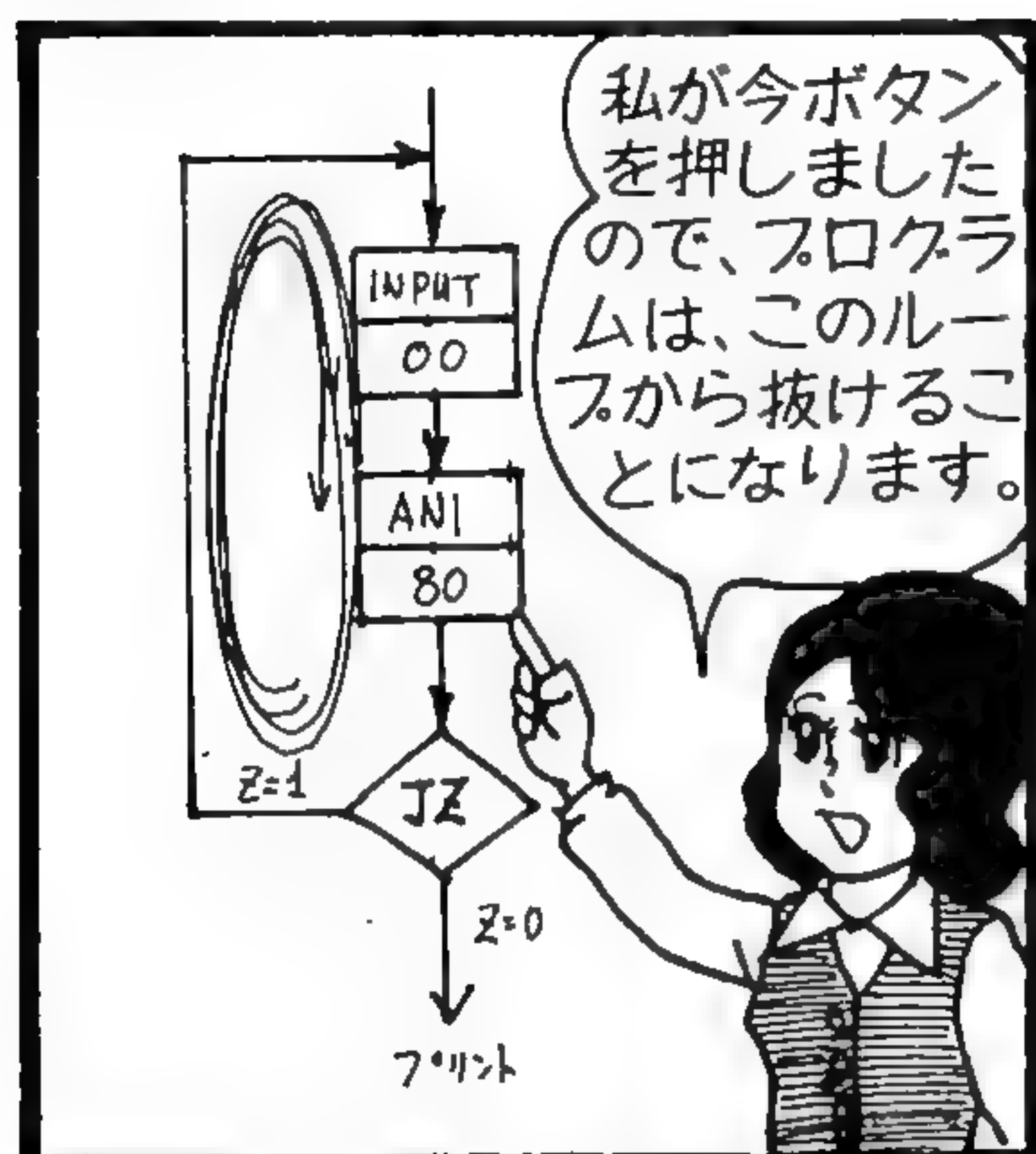
```

graph TD
    Start(( )) --> Input[INPUT 00]
    Input --> ANI[ANI 80]
    ANI --> JZ{JZ}
    JZ -- Z=1 --> Loop(( ))
    JZ -- Z=0 --> End(( ))
    
```

JZ...ゼロ  
ならジャン  
プ、Z=1  
ならばゼロ。  
くそ、やや  
こしい  
す。

フロントボタン  
が押されたか判断





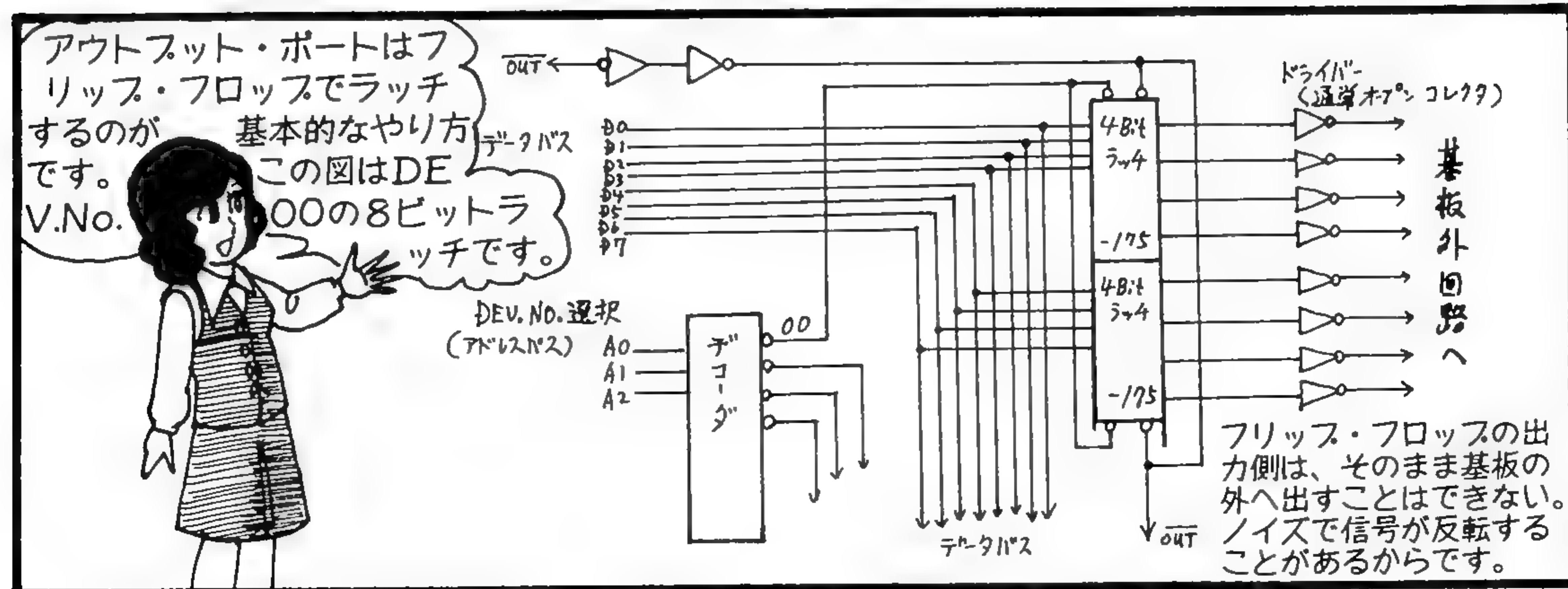
番号は INPUT と同じですがコントロール信号の INP OUT は同時に出ることはないから OK です。

あつ、そうそう。どこに何を出力するのか OUTPUT 表をつくっておかなくちゃいけませんね。

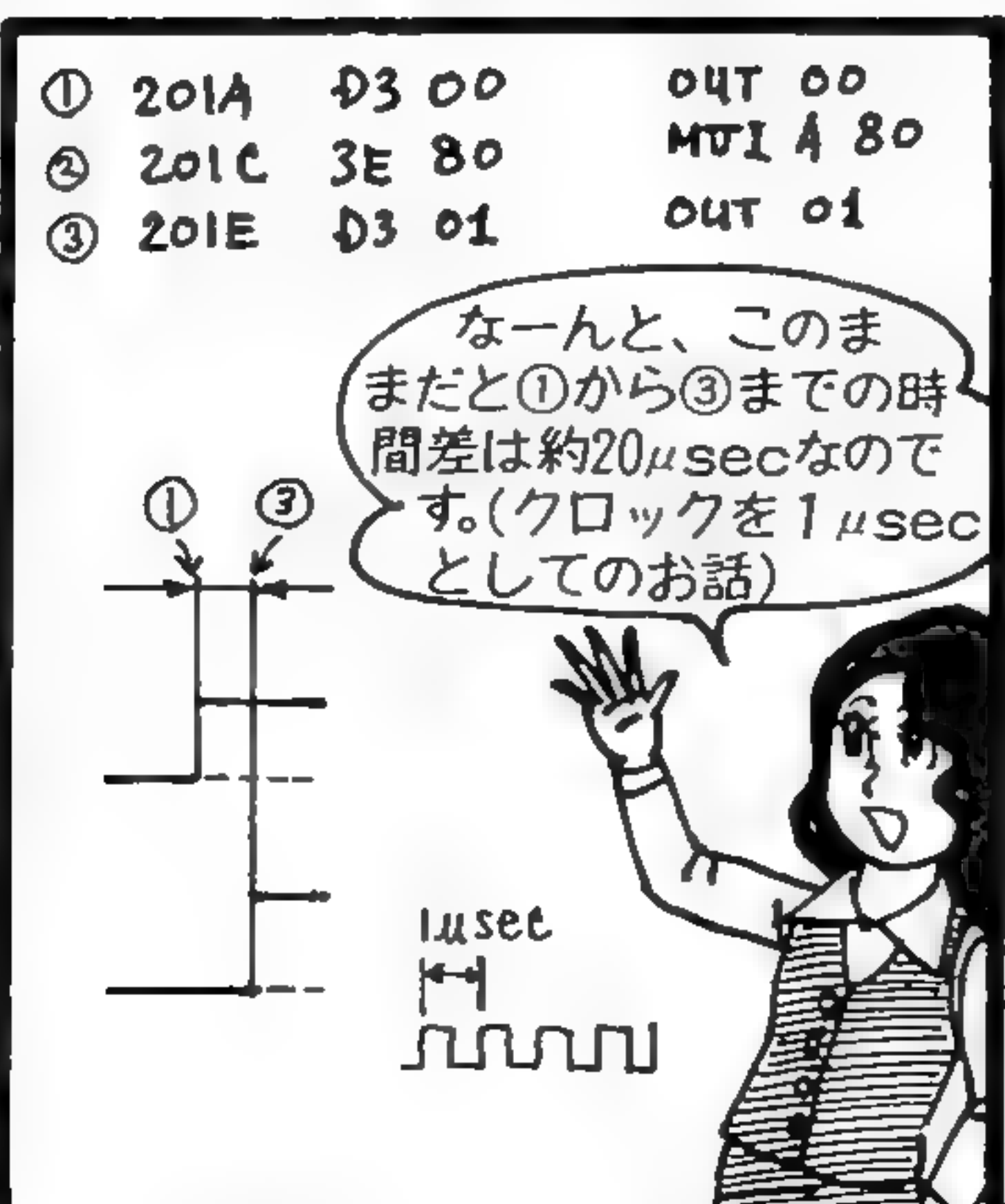
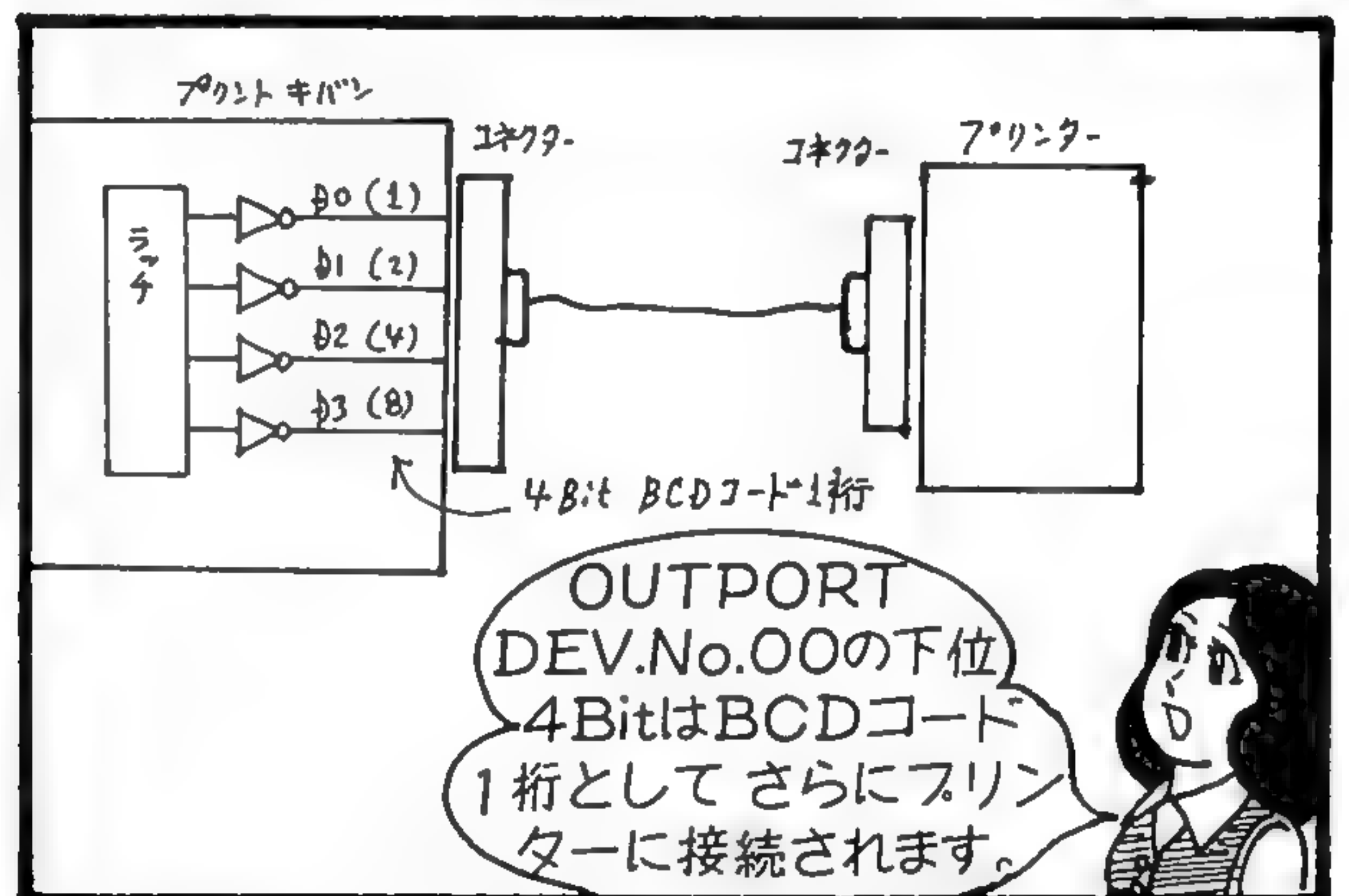
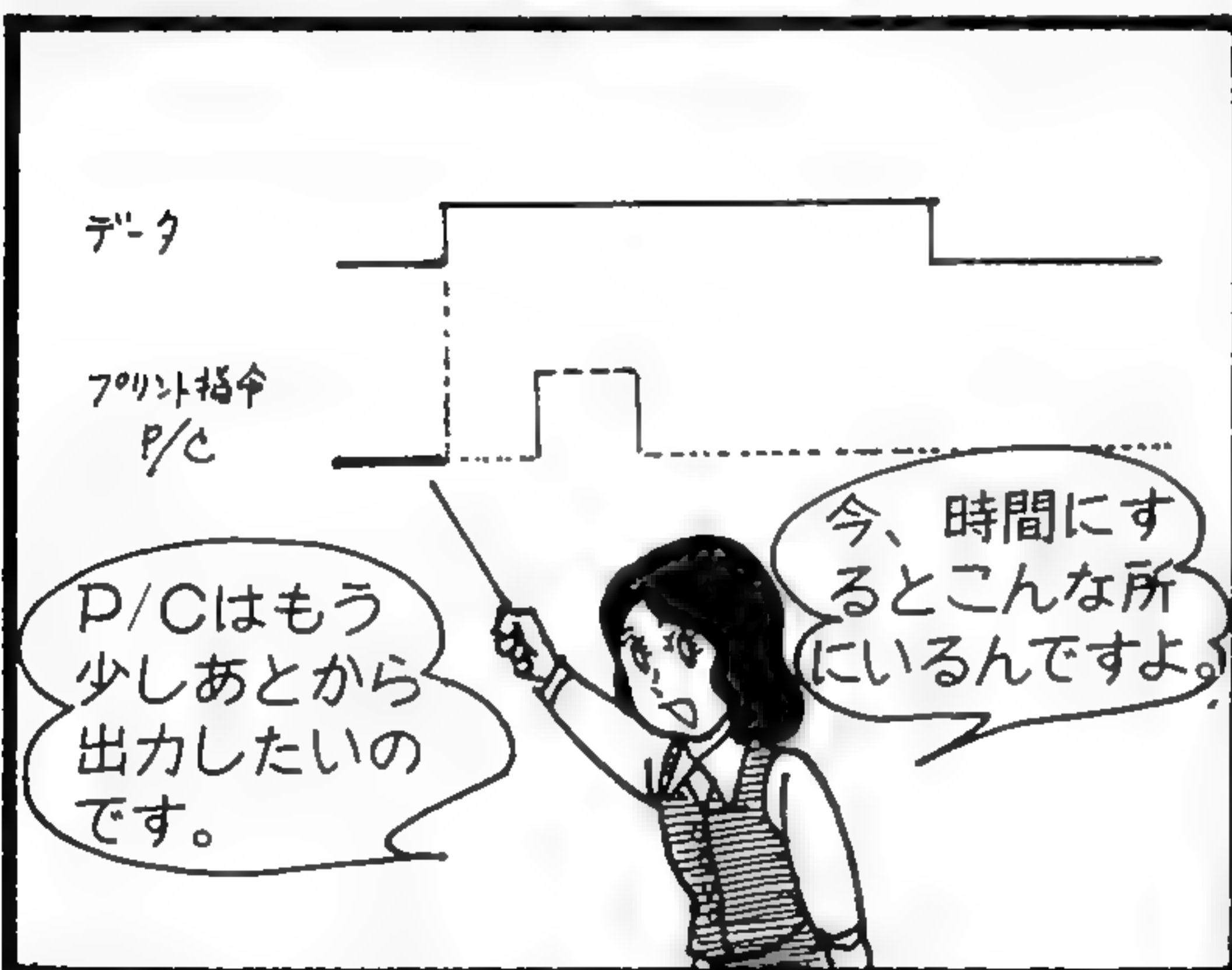
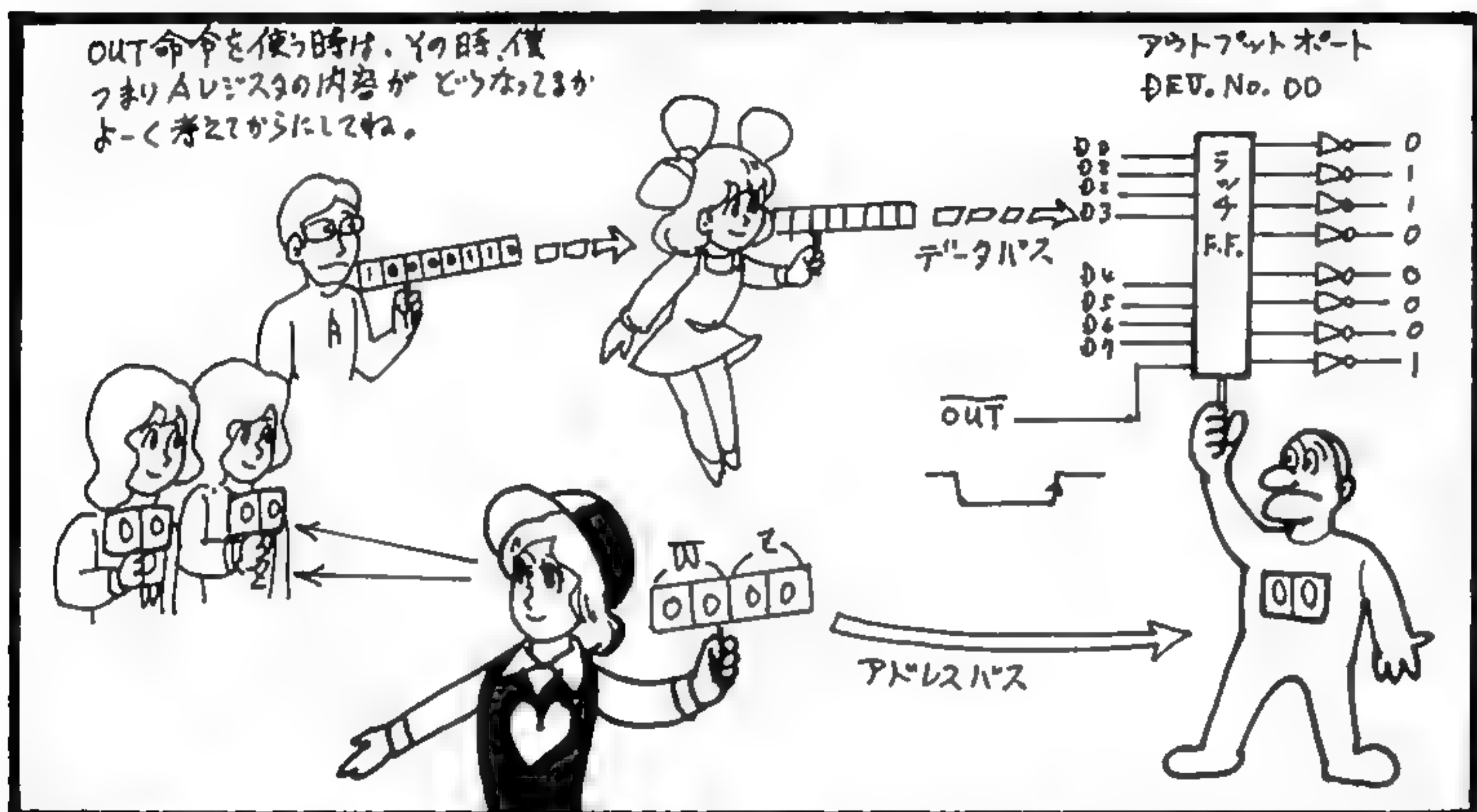
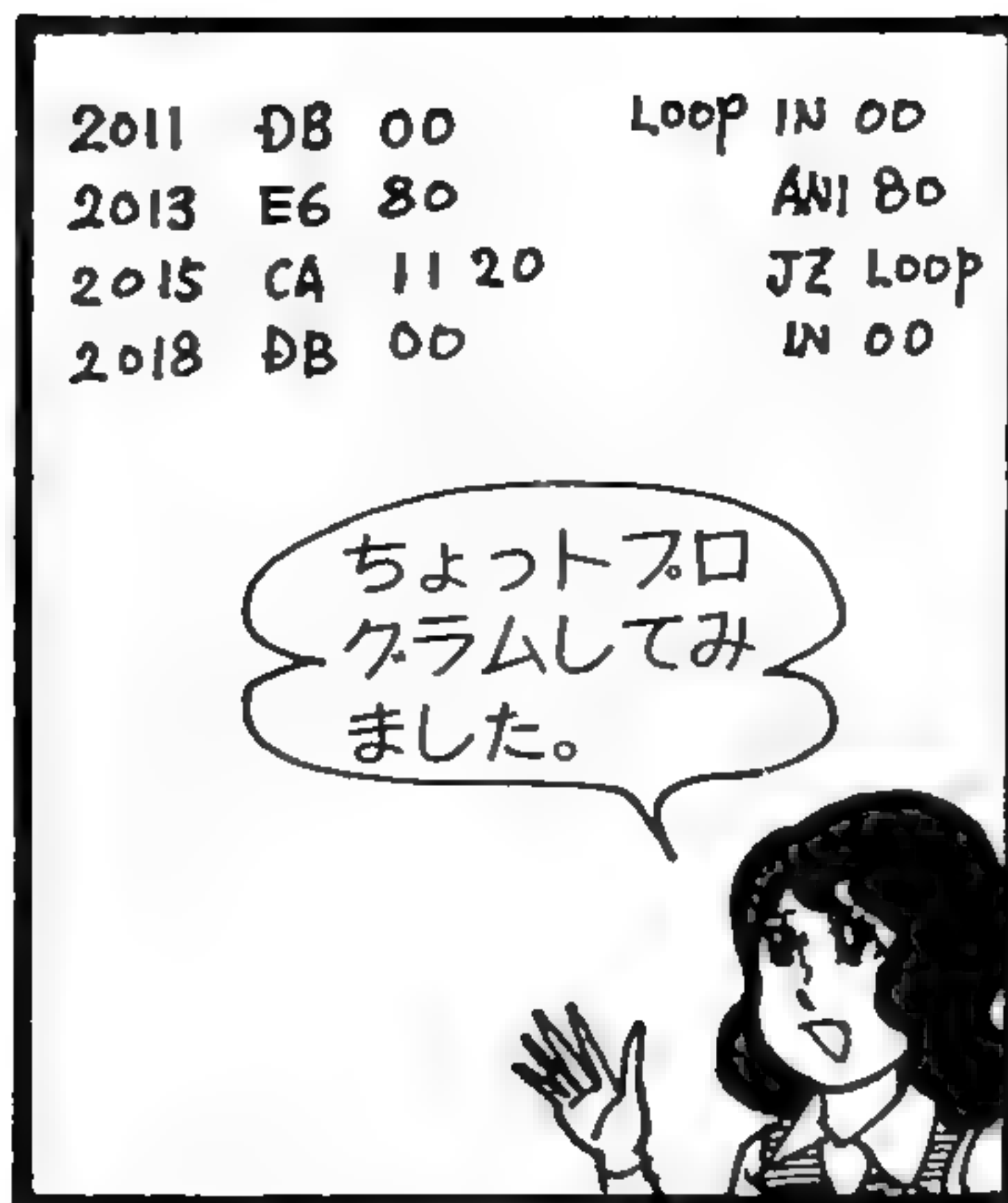
OUTPUT 表

DEV No	D7	D6	D5	D4	D3	D2	D1	D0
00					8	4	2	1
01								

プリントデータ 4 Bit  
プリント指令 P/C













ちょっとサービスして10進→16進への変換方法をお教えしておきましょう。まず、10進→2進の変換をします。

2000(D) → ? (B)

↑                      ↑  
デシマル              バイナリー  
(10進数の意)        (2進数の意)

そこでMVI B, 2000(D)となります。でも、ハンドアセンブルする時は、これは16進(H)に変換しなければなりません。

ずらせる時間なんてだいたいでいいのですから17÷20として、2,000回ぐるぐる回してやりましょう。キチンとしたことが好きな人は17として計算してください。

2) 2000  
2) 1000 ... 0 a  
2) 500 ... 0 b  
2) 250 ... 0 c  
2) 125 ... 0 d  
2) 62 ... 1 e  
2) 31 ... 0 f  
2) 15 ... 1 g  
2) 7 ... 1 h  
2) 3 ... 1 i  
1 k ... 1 j

で順序を示すためにつけたもの  
他の意味はありません。

もう割れないのでこれで終わりです

この要領で次々と割っていき余りをメモしていきます。

2) 2000  
2) 1000 ... 0  
2) 500 ... 0  
2) 250 ... 0  
2) 125 ... 0  
62 ... 1

ここで初めて余りが1と出ました。続けます。

これは10進の数字を順次2で割っていくことで得られます。まず2000を2で割ります。答は1000で余りはありませんので0とします。これは答の横にメモしておきます。

2) 2000  
1000 ... 0

↑                      ↑  
1回目の答          余り

11111010000 (B)  
7 D 0 (H)  
つまり...

2000(D)→7D0(H)となります。

校算

$$7 \times 16^2 = 1792$$

$$+ 1) D - 7 \text{ 割 } 13 \times 16 = 208$$

$$2000(D) \leftarrow \dots \text{あてゐる!}$$

2進→16進の変換はもっとかんたん。4ビットずつ区切って16進コードに直せばいいんです。

k j i h g f e d c b a  
1 1 1 1 1 0 1 0 0 0 0

これをkの方から順に並べると2000(D)を11111010000(B)に変換したことになるんです。

では、2000=40×50というぐあい分割してウェイト(待ち)時間のルーチンを考えていきましょう。

ポイント →

40

50

えへ、これは失敗です。Bレジスタは8ビット、つまり256(D)しか扱えないのでありましたっけ。

MVI B, FF(H)

↓  
255(D)

が限界なの

MVI B, 7D0 ?!

そんなま...



		レジスタの内容
1	MVI B, 02 (B=2)	
2	MVI C, 02 (C=2)	
3	DCR C (C=2-1=1)	
4	JNZ T1 (C≠0) T1にジャンプする	
5	DCR C (C=1-1=0)	
6	JNZ T1 (C=0) T1へジャンプしない で次へ	
7	DCR B (B=2-1=1)	
8	JNZ T2 (B≠0)	
9	MVI C, 02 (C=02)	
10	DCR C (C=2-1=1)	
11	JNZ T1 (C≠0)	
12	DCR C (C=C-1=0)	
13	JNZ T1 (C=0)	
14	DCR B (B=1-1=0)	
15	JNZ (B=0) ここで抜ける。	

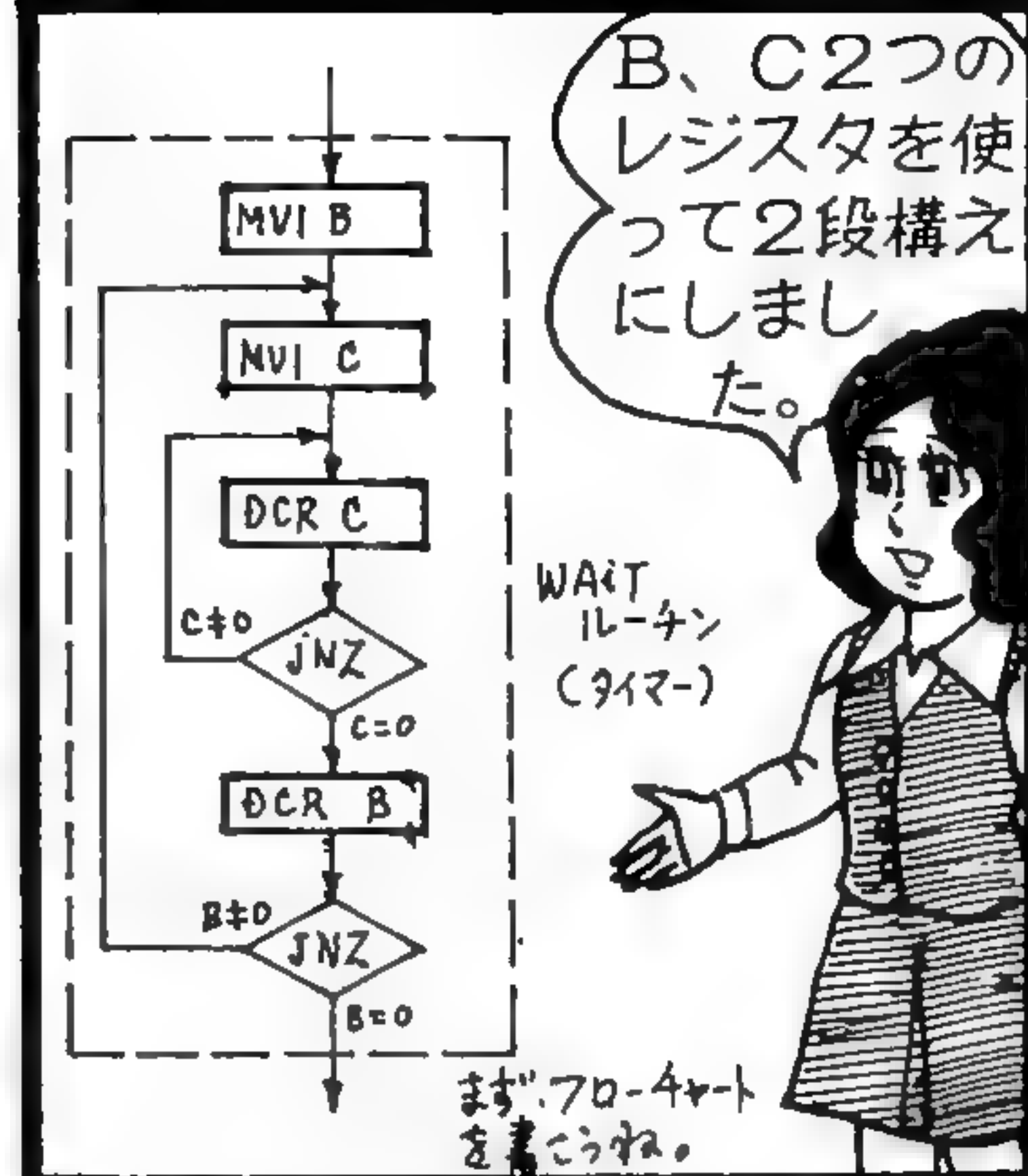
前のコマのプログラムをCPUが実行すると、こんなぐあいになるんです。

まず最初わかりやすいようにB、Cそれぞれに2を入れた時どうなるか、一般的に成立する式を導いてそれから $2000=40 \times 50$ の場合を考えることにします。フローチャートをプログラムすると……

```

MUX B, 02H
T2: MVI C, 02H
T1: DCR C
    JNZ T1
    DCR B
    JNZ T2
    :

```



WAIT  
IL-4ン  
(タイマ-)

まず、70-4を  
書きこうね。

これを見て  
関係式を導  
きます。

レジスタの内容  
↓                  ↓  
レジスタの内容

DEC JNZ } ... (B) × ((C) + 1) = 2 × 3 = 6 ⑫

MVI B, 02 ... = 1 ⑬

MVI C, 02 ... (B) = 2 ⑭

ホントかなあ、と思う人は $B=3$ 、 $C=2$ または $B=2$ 、 $C=3$ として演習問題のつもりでやってください。

DCR } の組は17ステート、MVI、r、D8の組は  
JNZ }  
7ステートです。そこで、1ステート1 $\mu$ secと  
すると、  
$$(2040 \times 17 + 40 \times 7 + 1 \times 7) \times 1 \mu\text{sec} = 34967$$
  
$$= 34967 \mu\text{sec} .$$
  
となります。目標は40msec、これは35msec  
ですからだいたいいい線でしょう。  
これじゃ気持ちの悪い人はB、Cにもっと40msec  
に近くなるものを入れてくださいね。

関係式ができましたので、  
 $B=40$ 、 $C=50$ を入れて時間  
 を計算することにします。

実行回数

DEC R } ... (40) × (50 + 1) = 2040 回

INC R }

MVI B, 40 (D) = 1 回

MVI C, 50 (D) ... (B) = 40 回

↑  
レジスタの内容

サブルーチンのラベル  
、（最初のOPコードの番地）

```

WAIT: MVI B, 28H
      T2: MVI C, 32H
      T1: DCR C
           JNZ T1
           DCR B
           JNZ T2
           RET

```

リターン文をつける。これは10ステート必要。気になる人はこれも含めてください。 $34,967 + 10 = 34,977 \mu\text{sec}$

これで35ミリ秒の  
タイマーができた  
ことになります。  
何度も使えるよ  
うにサブルーチ  
ンにしておき  
ましょう。

$$\begin{array}{r} 2 \overline{) 40} \\ 2 \overline{) 20} \dots 0 \\ 2 \overline{) 10} \dots 0 \\ 2 \overline{) 5} \dots 0 \\ 2 \overline{) 2} \dots 1 \\ 1 \dots 0 \end{array}$$
$$\begin{array}{r} 2 \overline{) 50} \\ 2 \overline{) 25} \dots 0 \text{ a} \\ 2 \overline{) 12} \dots 1 \text{ b} \\ 2 \overline{) 6} \dots 0 \text{ c} \\ 2 \overline{) 3} \dots 0 \text{ d} \\ 1 \dots 1 \text{ e} \end{array}$$

40と50を  
16進に直  
しておき  
ましょ  
う。

$$\underbrace{101000}_2 \quad \underbrace{110010}_3 \quad \begin{matrix} f & d & c & b & a \\ 1 & 1 & 0 & 0 & 1 & 0 \end{matrix}$$
$$\therefore 40(D) = 28(H)$$
$$50(D) = 32(H)$$

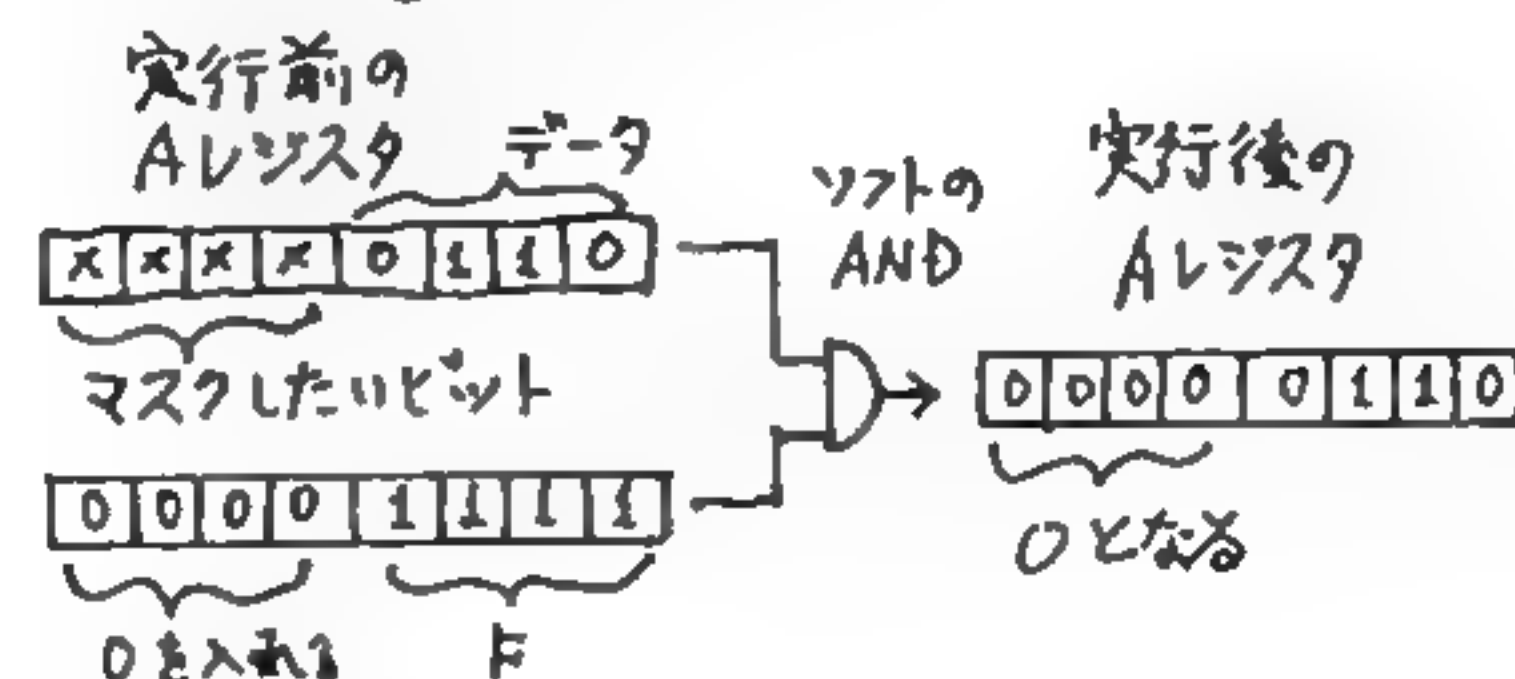


メイン  
プログラム

```

Loop: IN 00 : 00ポートよりデータ読み込み
      ANI 80 : } P/C (プリントコマンド) 押金の4エック
      JZ Loop : } 押されるまでここをフルクル回っている。
      IN 00 : 00ポートのデータを改めて読み込み
      ANI 0F : 上位4ビットをマスクする。

```



```

OUT 00 : データをOUT PUT
CALL WAIT : 35 msec待つ
MVI A, 80 : P/Cビット (P/C) をセットする
OUT 02 : P/C をOUT PUT
CALL WAIT : 35 msec待つ
MVI A, 00 : P/Cのリセットをセット
OUT 02 : P/C をリセット
CALL WAIT : } 35x3=105 msec + α 待つ
CALL WAIT :
CALL WAIT :
OUT 00 :

```

データをリセット  
(Aレジスタは今00が入っている)

サブルーチン

```

WAIT: PUSH B
      MVI B, 28H
T2: MVI C, 32H
T1: DCR C
    JNZ T1
    DCR B
    JNZ T2
    POP B
    RET

```

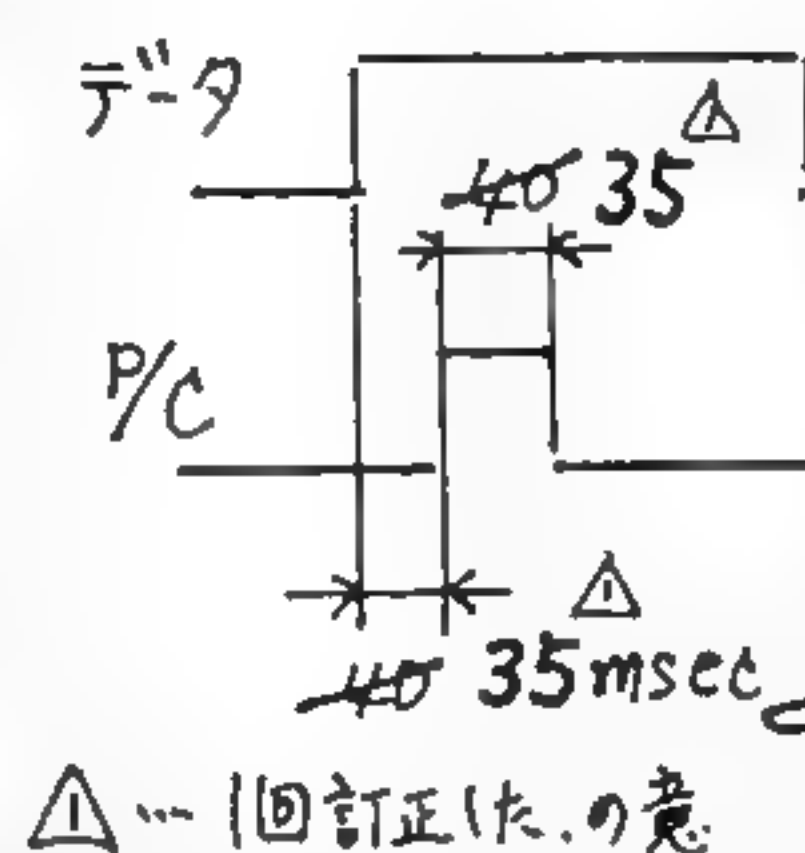
35 msec WAITの  
サブルーチン

## &lt;問題&gt;

LOOP → 2000 番地  
WAIT → 2100 "

として、ハンドアセンブリ・プログラムリスト  
を作成してください。

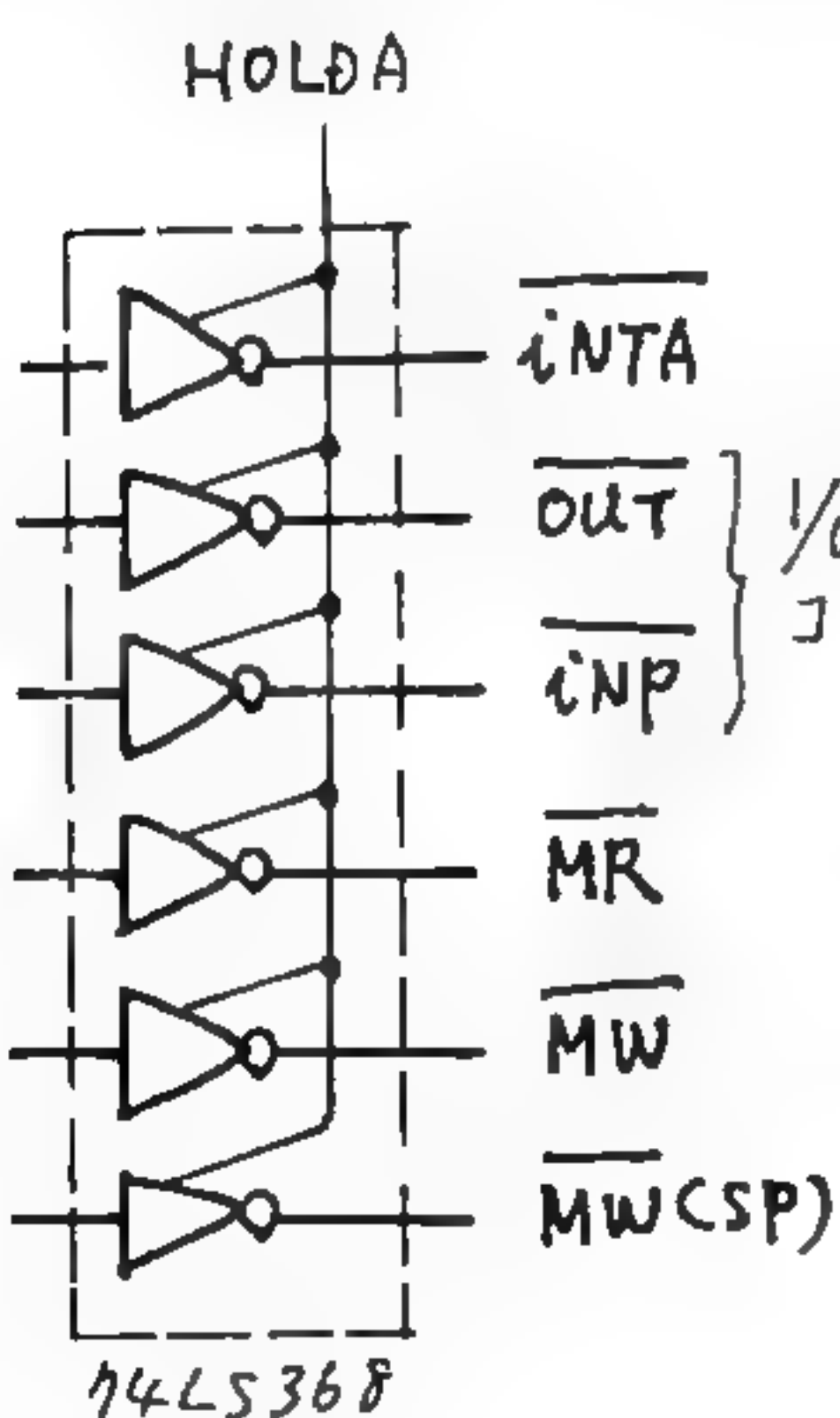
私だったら  
タイム・チャ  
ートの方を修  
正しちゃいま  
す。



自分でモデルを想  
定して書いてみると、  
よくわかりますよ。  
とかく、人の書い  
たのは、先入観もあっ  
てよくわからないもの  
です。

入出力のプログ  
ラムを整理する  
と、このよう  
になります。

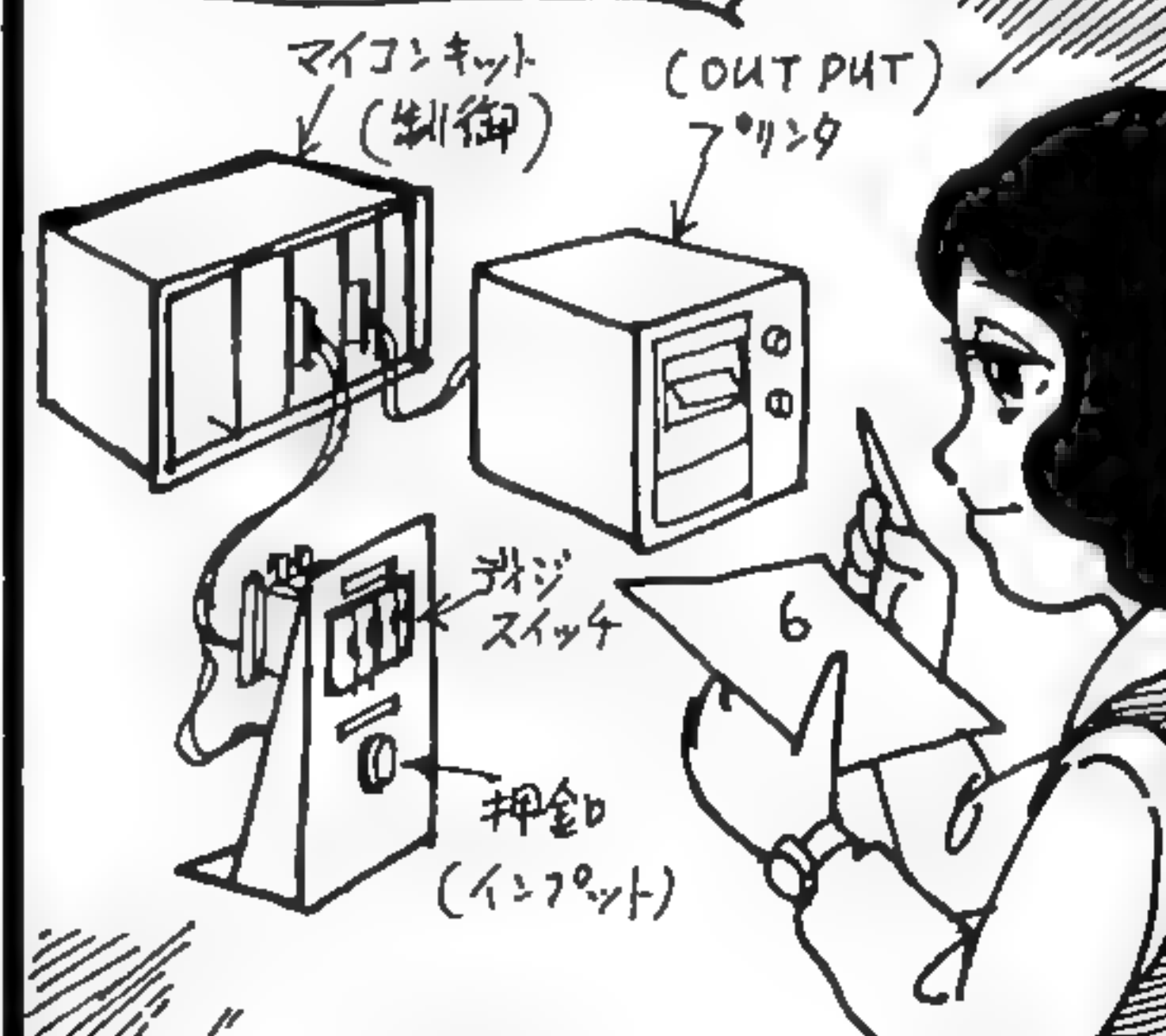
マスクとい  
う言葉の感  
じをつかん  
でくださ  
い。



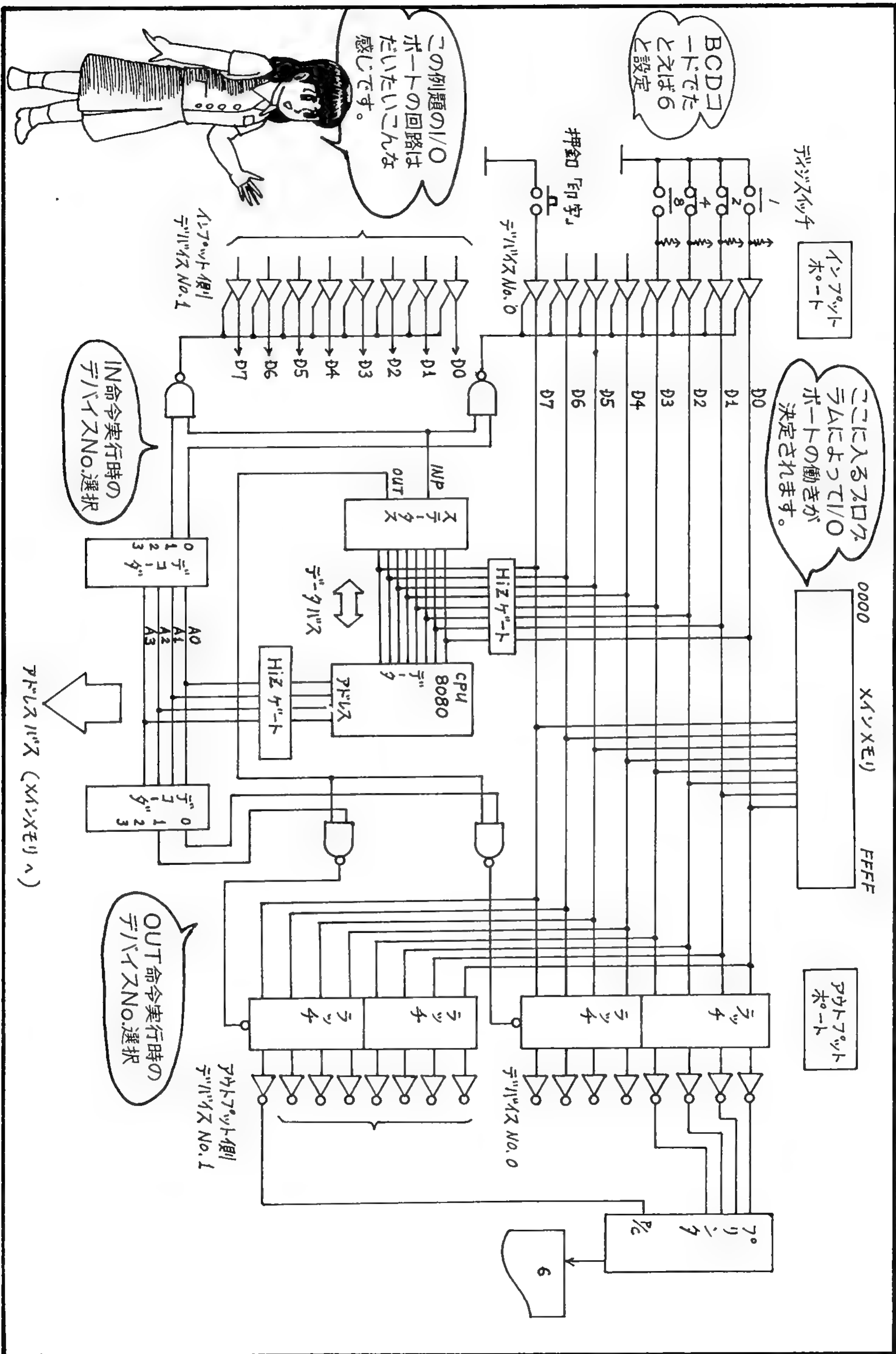
I/Oポートのコントロールはステ  
ータスインフォメーション(S.I.)  
のデータバスのビットD4、D6を  
使います。これでコントロール・  
バスのすべてが明らかになり  
ました。



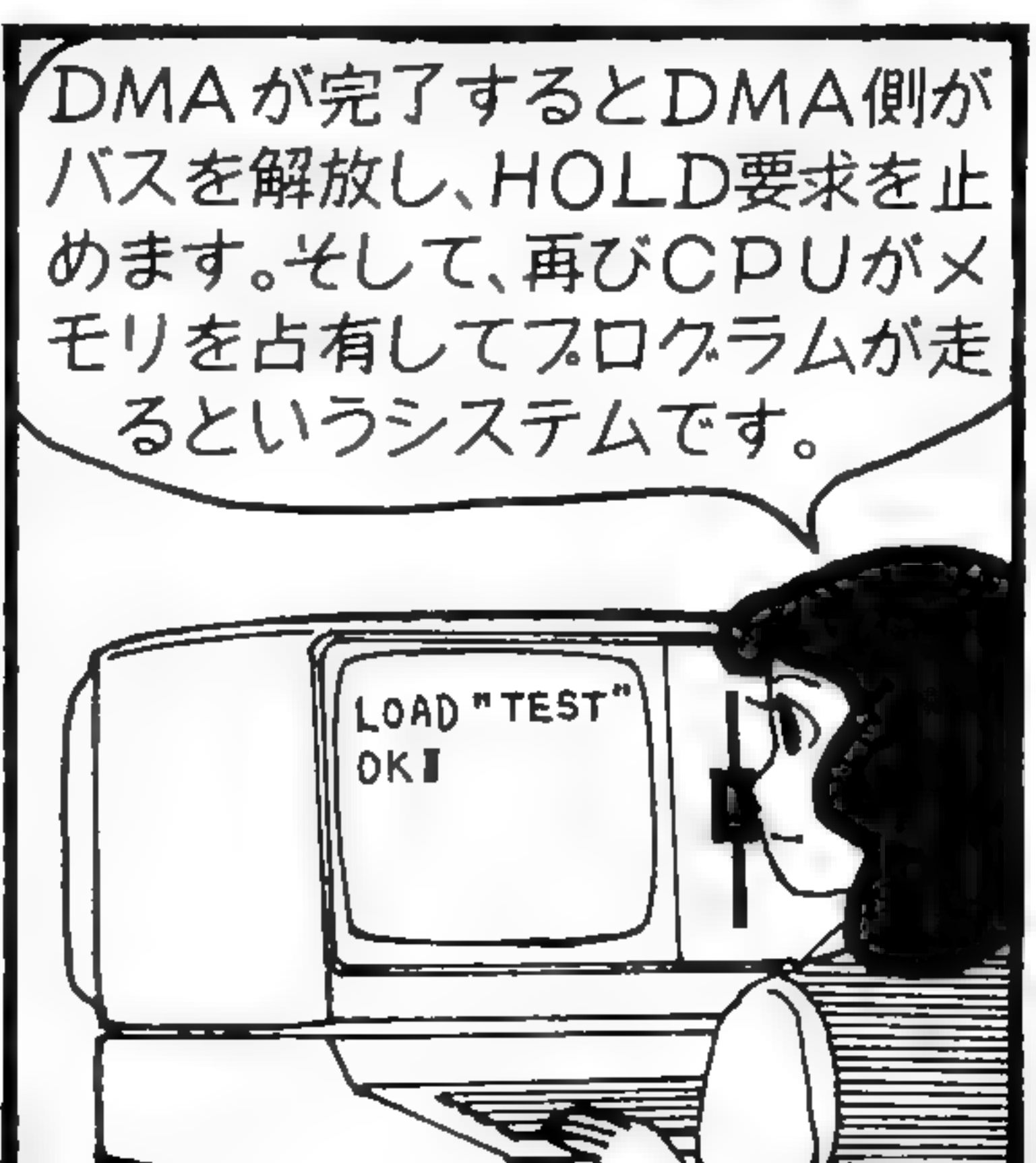
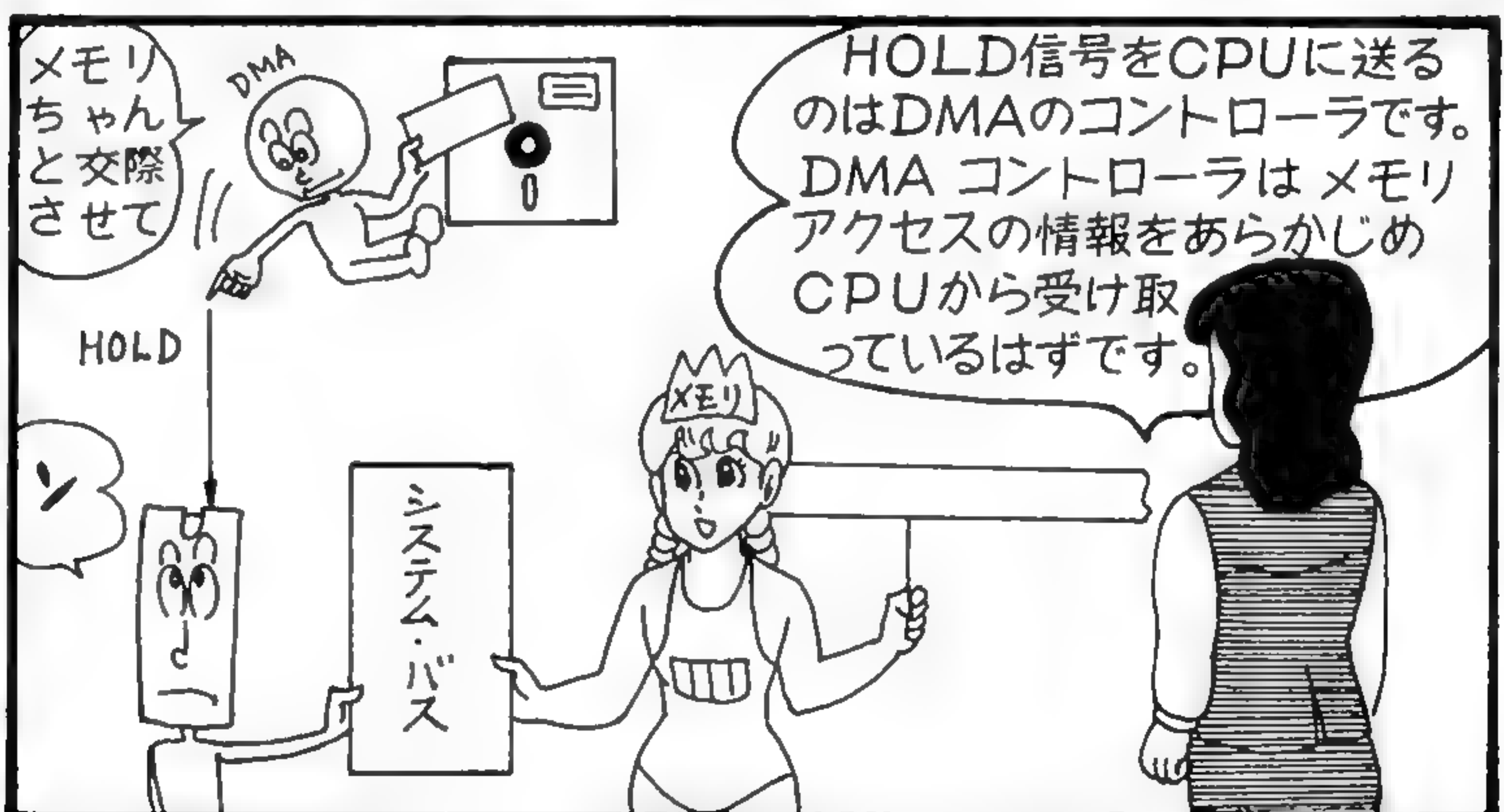
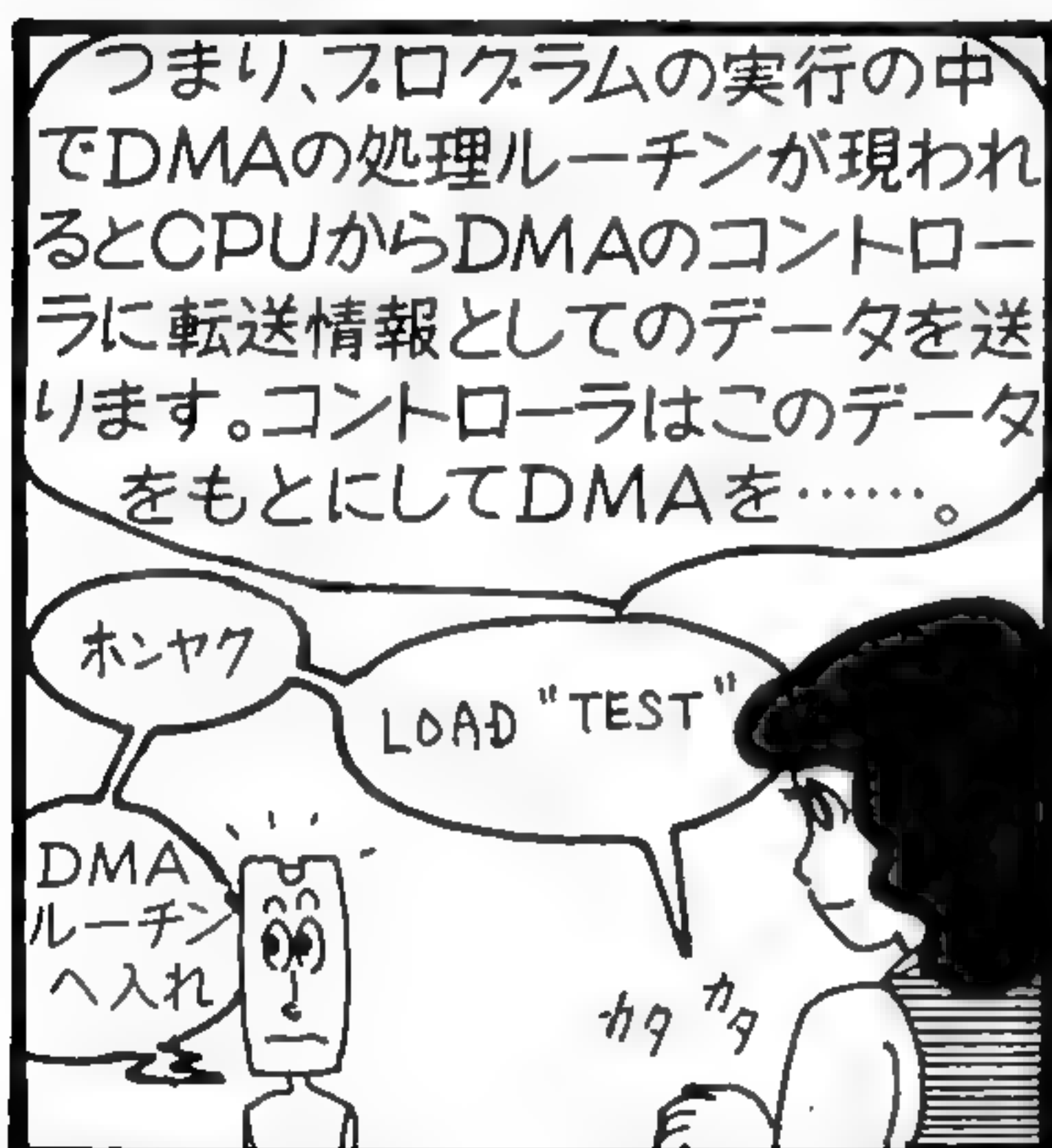
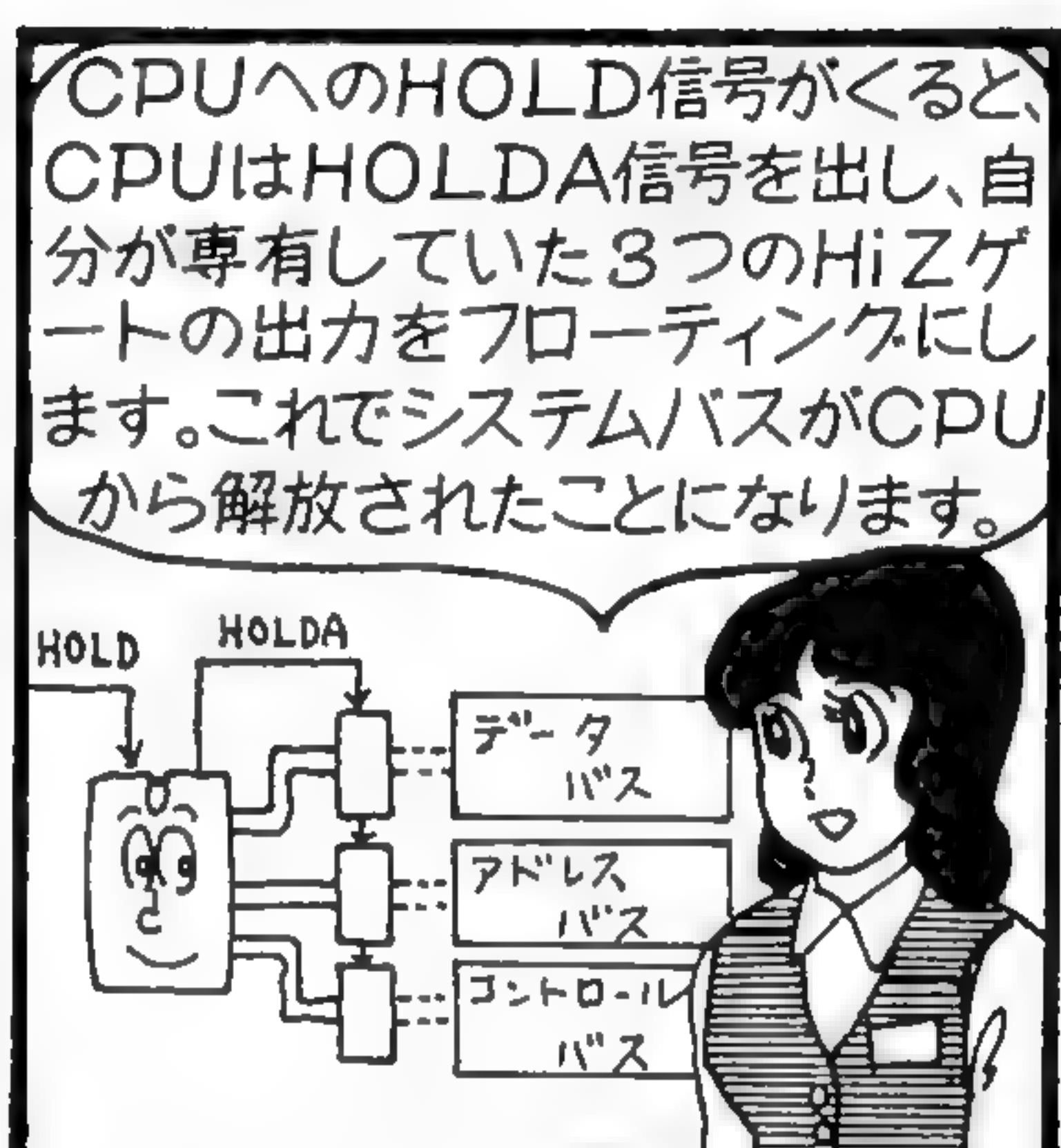
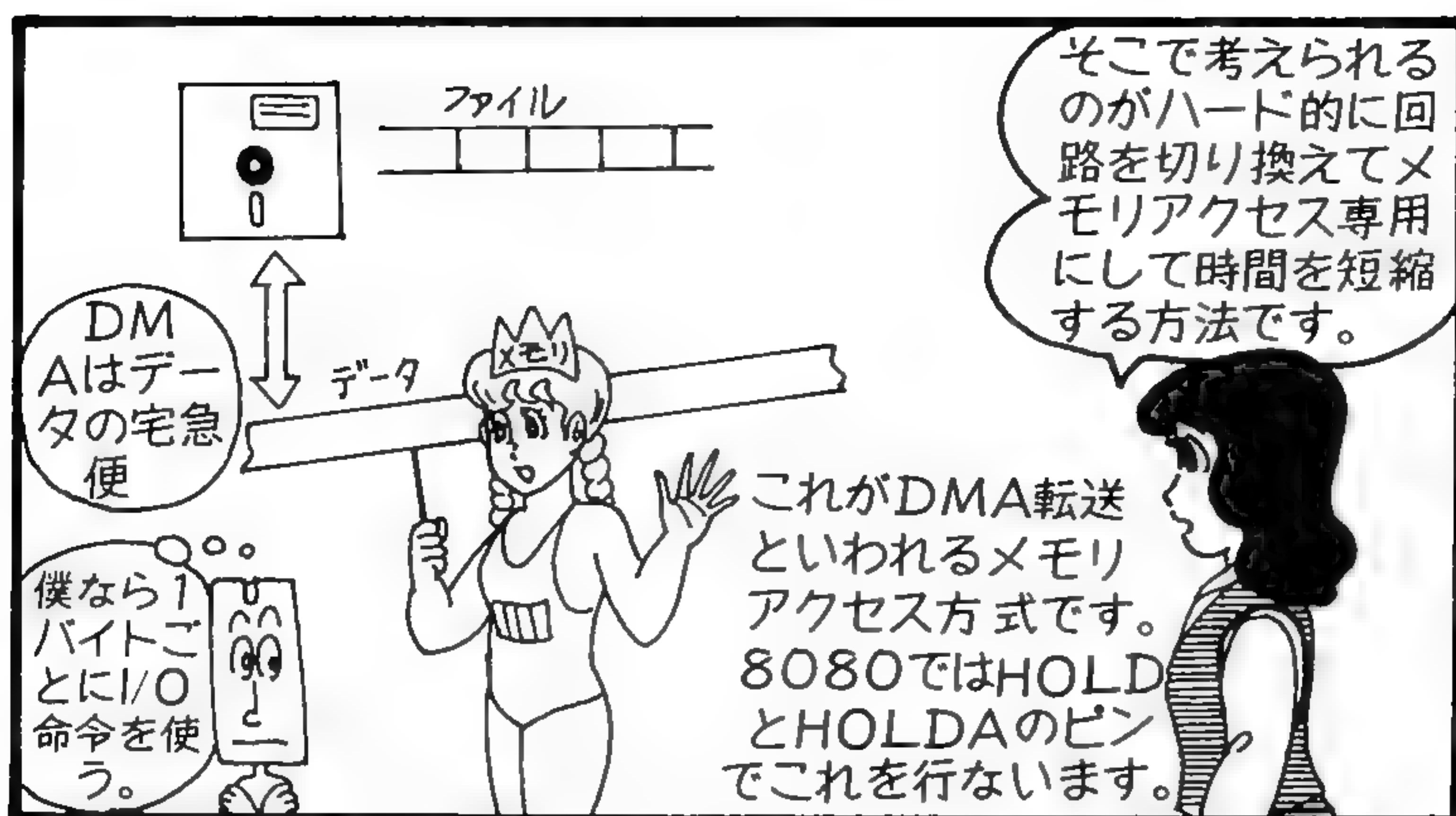
これでディジスイッチの数字1  
桁をCPUを通してプリントす  
るシステムができたわけです。



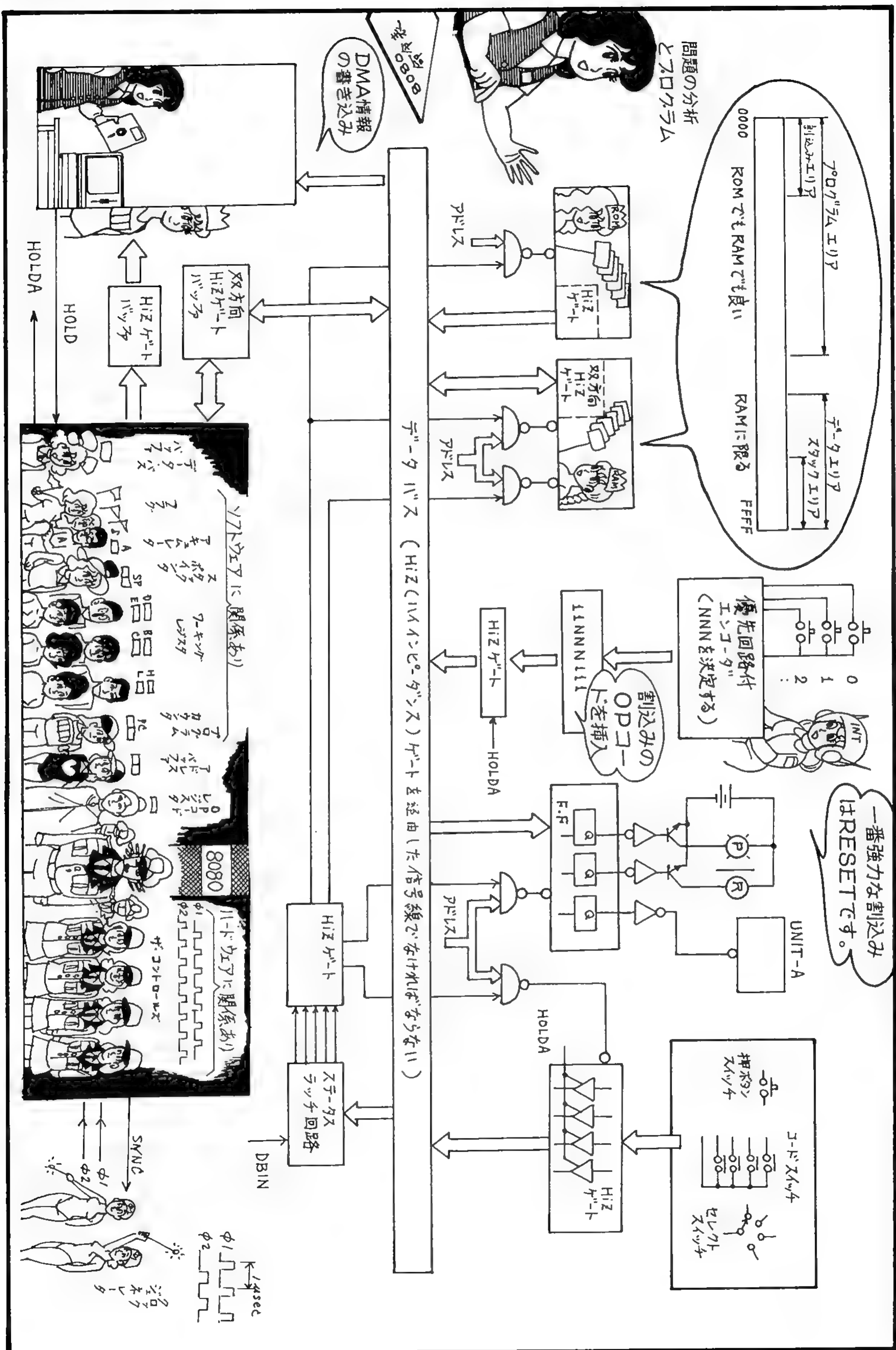










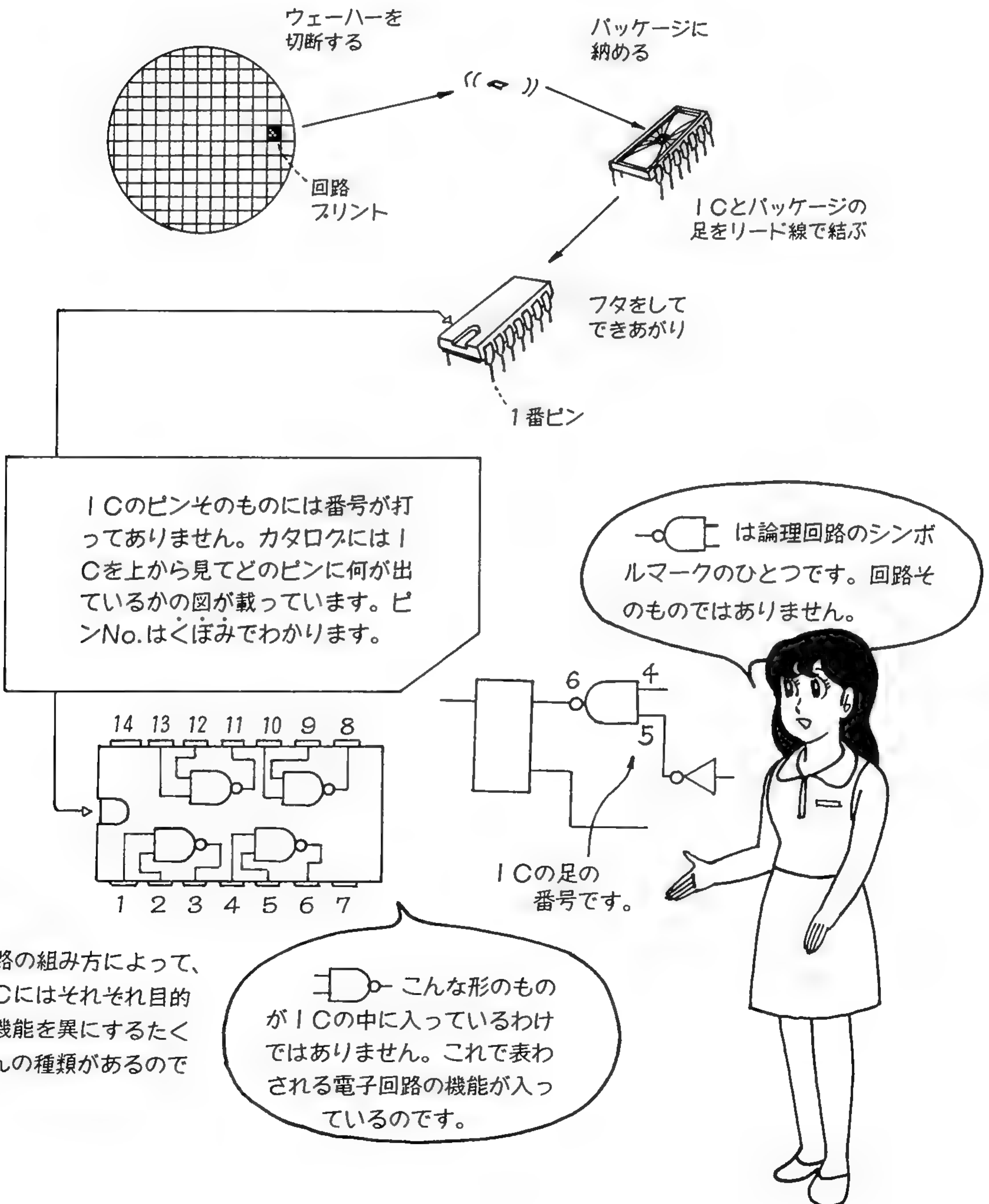




## ICのQ&A

\* ICについてもう少し話してください

ICというのは一般にムカデのような形をしています。本当はあれはICを入れるプラスチックのケースなのです。そしてムカデのような足はプリント基板やソケットに入れやすいようになっているのです。IC自身は小さくて薄いものです。ICはせんべいのような丸いウェーハーにたくさんの同一回路を「写真印刷」して作られます。そして切断してひとつひとつの回路に分けパッケージの中に納めるのです。





## □ CP/Mとアセンブラ の活用知識

BASICを走らせるためには、BASICインタプリタが必要です。モニターで書いたプログラムを走らせるためにはアセンブラが必要です。

では、このインタプリタやアセンブラを作るためにはどうしたらいいのでしょうか。

このようなプログラムを作ることをソフト開発と呼んでいますが、このソフト開発用に作られたのが、ディスクオペレーティングシステムを持つCP/Mという8080用のソフト開発システムなのです。

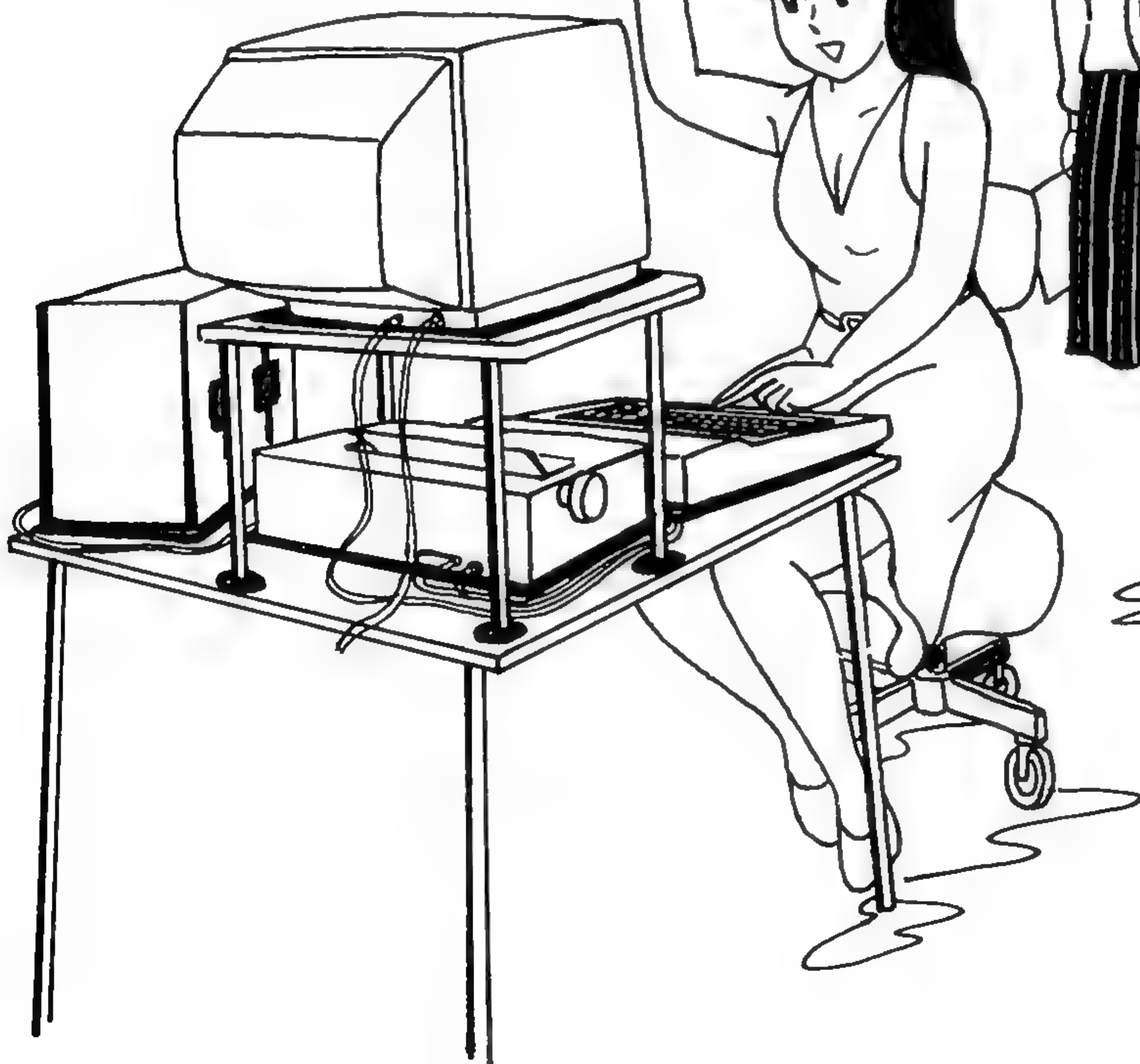
CP/Mは8ビットパソコンのOSとしてオールRAM型パソコンに採用されつつあります。

使用者の能力に応じて  
フロからホームユーザ  
ーまで。

ゲームで遊  
ぼう。

宿題の答を  
BASICで検算  
するの。

私はマシン語の  
勉強中。

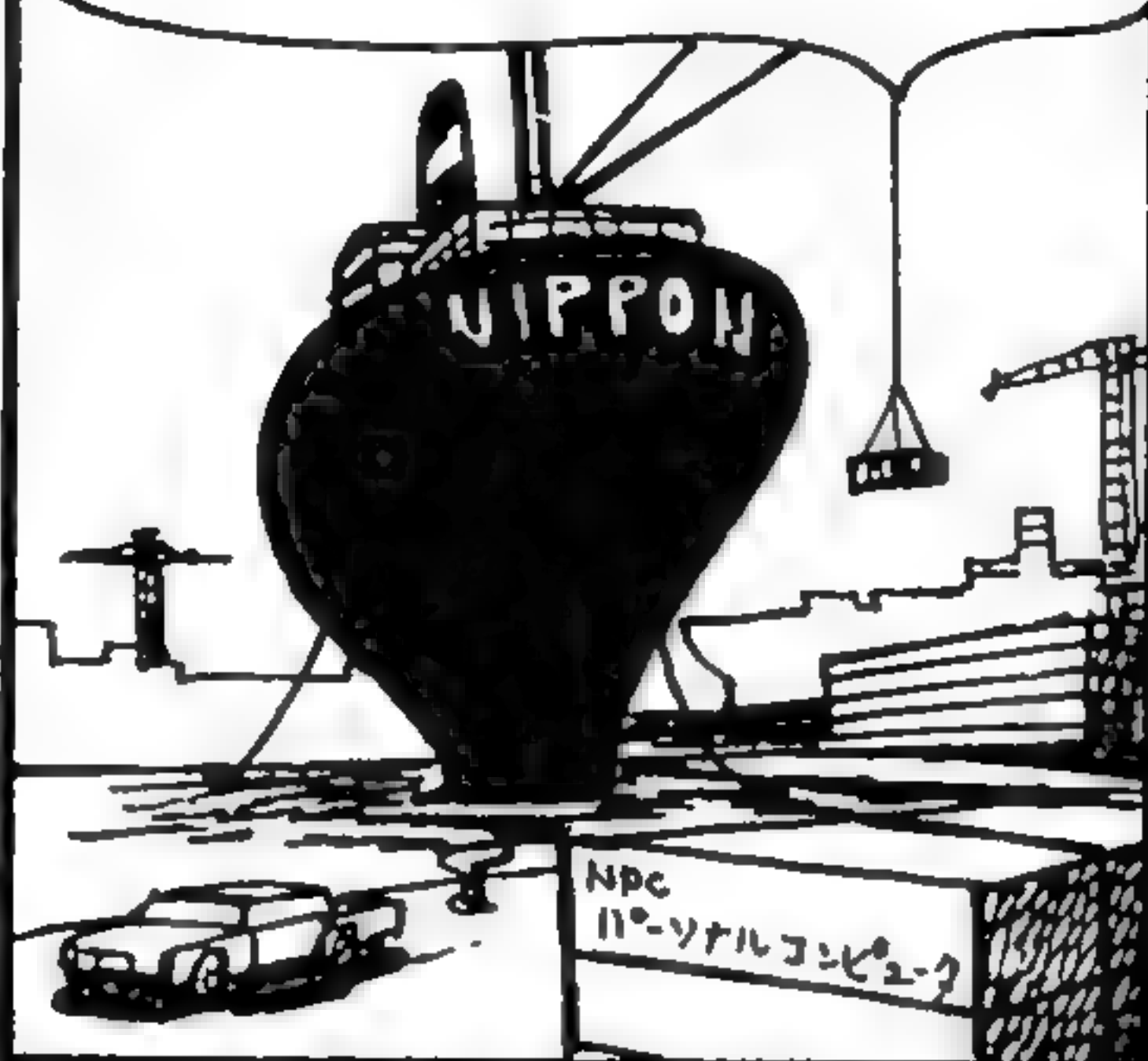




メモリなどの、ICレベルではシリコンバレーのIC業界と日本の業界で多少の摩擦があるようですが、パソコンのレベルではまだそれがありません。



日本はいろんな工業製品をアメリカなどに輸出していますが、パソコンではボツボツ輸出しているという程度なのです。



## ① CP/Mとは何か



今8ビットではオールRAM  
の上でCP/M というプロク  
ラムが走らないパソコンはパ  
ソコンでないというのがあち  
らではほとんど常識なのです。



CD・Mは  
0番と1番の  
2つのトラッ  
クの中に入っ  
ています。

なぜ売れないかというと、日本のパソコンはROM型で、BASICしか走らないものが多いからです。



オーノー、  
コレ BASIC  
オンリー？  
ワタシ、  
買イマセーン。

なぜでしょうか？ それは今の日本のパソコンを輸出しても現地ではそう売れないからです。



オー、  
ニホンノ  
パソコン  
デスカ？

O BASIC(OKI BASIC)

(ED(エディタ))

ASM(8080アセンブラ)

LOAD(ローダ)

(WMIF(ワードマスター))

M80(Z80アセンブラ)

L80(ロータ)

(WMIF(ワードマスター))

PASCAL MT+(パスカルコンパイラ)

MTPLUS 000

オーバーレイ用  
プログラム

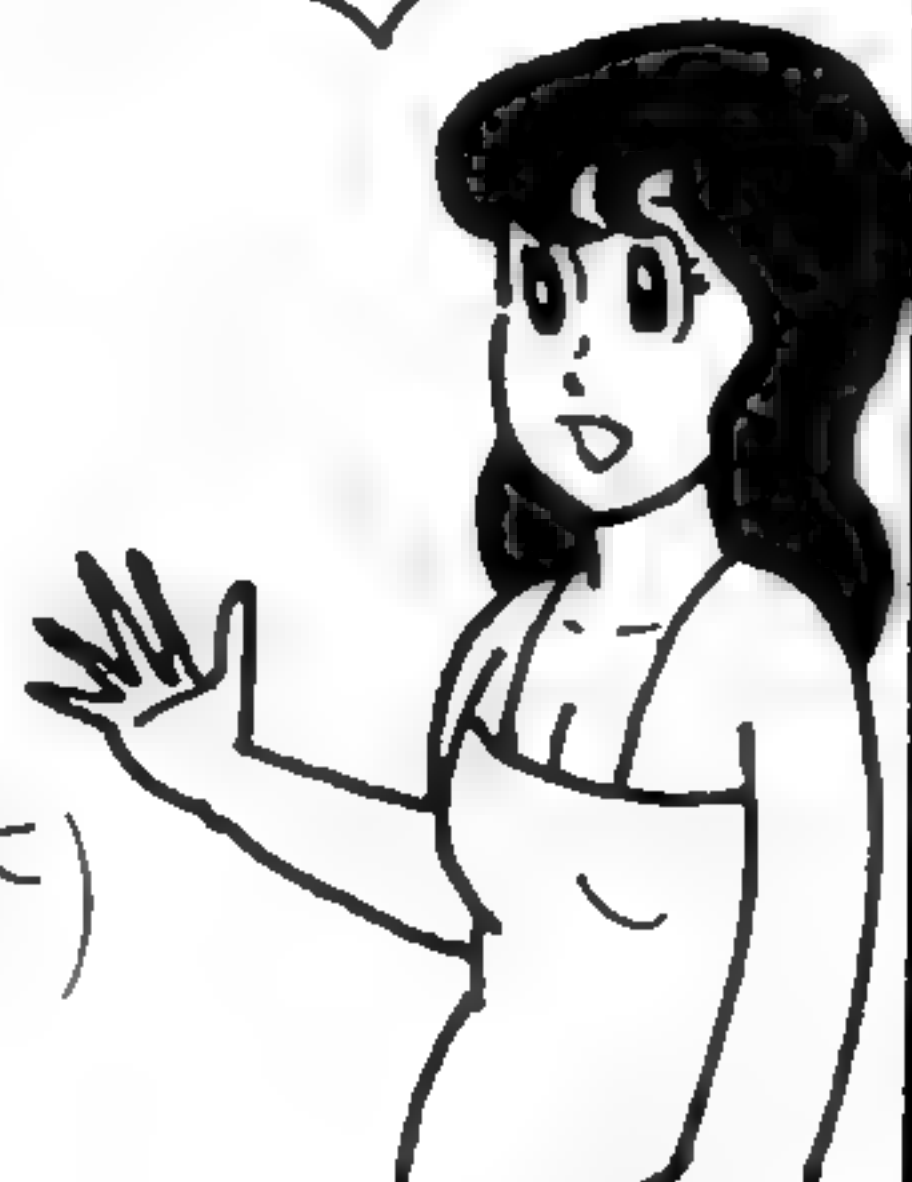
MI PLUS 006  
DASLIB(パ7カ)

LINKMT(リンカー)

17-ト

•

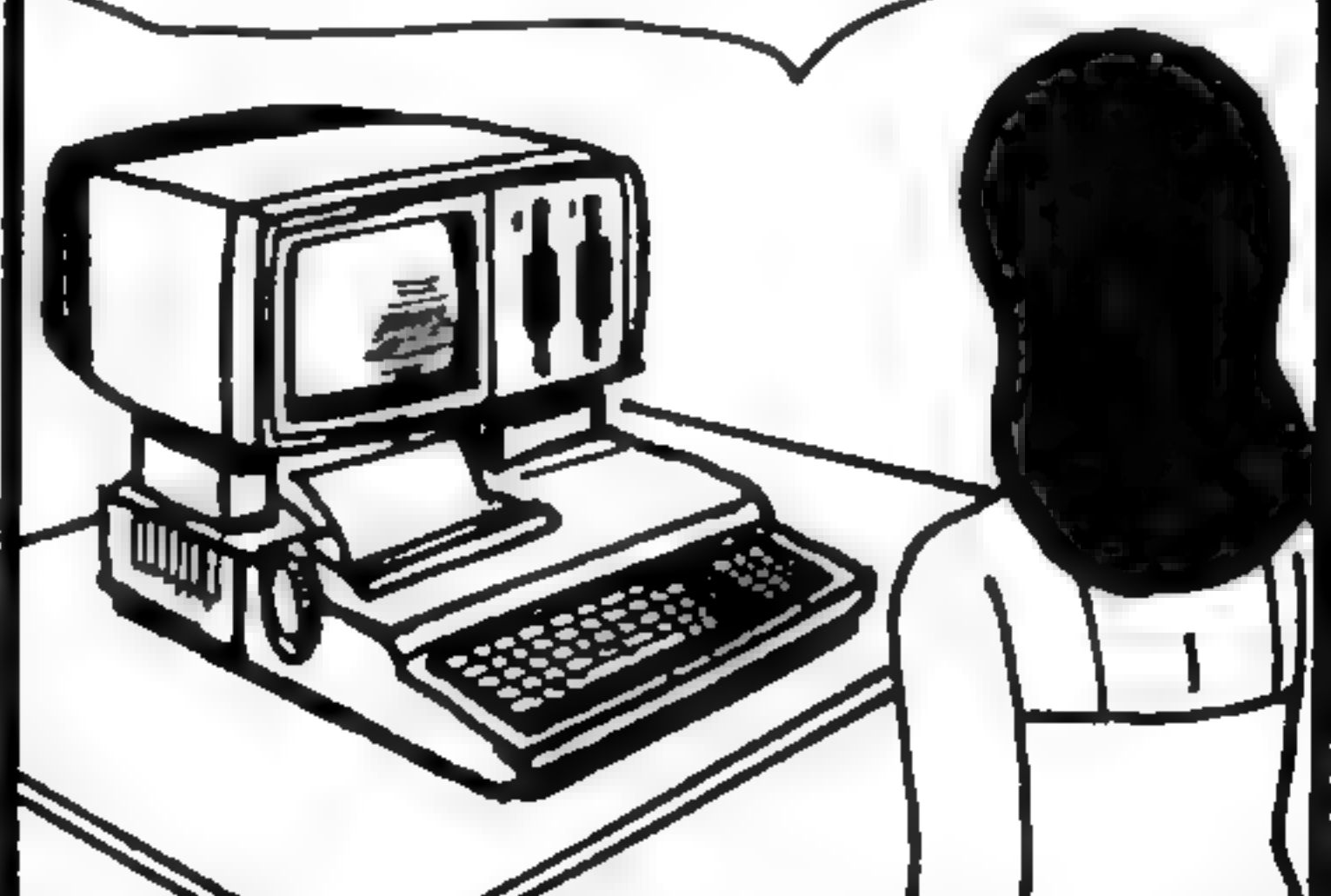
(ワードマスターはEDよりも楽に  
ソースファイルをつくれます。)



このパソコンではBASICは  
もちろんのこと、このような  
言語も走るのです。

BASIC、ED、ASM、LOADはパソコンを買った時の標準システムの中に入っています。

ここで、CP/M をくわしく  
お話するわけにもいきません  
が、これから使うパソコンの  
1F800-30 (沖電気) がた  
またまこの CP/M を使って  
いるので、お話していく途中  
でだいたいどんなものかわか  
ってもらえると思います。





## CP/MのQ&A

### \* CP/Mって要するに何ですか

一言でいうとディスクオペレーティングシステム用の‘ソフト’です。そして、そのソフトは8080のマシン語で作られています。BASIC言語を使うためにはBASICインタプリタというマシン語プログラムが必要ですが、そのプログラムを作る時にこのCP/Mを使うと非常に効率的だということです。

- (1) CP/Mの‘本体’はディスク2トラック分だけで非常にコンパクトである。
- (2) PIP (ピップ) コマンドでフロッピーやRS-232Cなどへの入出力が簡単にできる。
- (3) システムコールと呼ばれるサブルーチン群があり間接的にI/Oをハンドリングするのでプログラム開発中はI/Oの仕様を気にすることはない。

CP/Mの走るパソコンのシステムディスクにはED (エディタ)、ASM (アセンブラ)、LOAD (ローダ)、DDT (デバッガ) が標準的についていますが、これは本来CP/Mがマシン語開発用のソフトだからです。BASICや簡易言語のプログラムを走らせるだけでは非常にもったいない感じです。

CP/Mが使えれば、そのソフト、ファイルの互換性が保たれるので、急速に普及したOSです。

### \* CP/Mのうえを走るものは

拡張子が.COMファイル (本当にマシン語になっているもの、ただファイル名に.COMを勝手につけてもダメ) のものはファイル名で即走ります。

BASICインタプリタ、ED、ASM、LOAD、DDT、PIPなどはすべて.COM形式のファイルとしてディスクの中に入っています。

CP/Mで使えるソフトは、ハードの制約を越えて、どのハードでも使える、というのがうたい文句です。また、CP/M上ではBASIC以外の高級言語、たとえばCOBOLやFORTRANなども使えるのも魅力です。

### \* CP/Mのハード環境条件

- (1) 2ドライブのディスクがあること。
- (2) オールRAM型のパソコンであること。
- (3) CRTがあること。
- (4) プリンタがあること。

初心者はひとつのシステムとして同じメーカーのものを使った方が、メーカーの情報サービスセンターのアドバイスを受けられて、無難です。

CP/M (Control Program for Micro-computer)

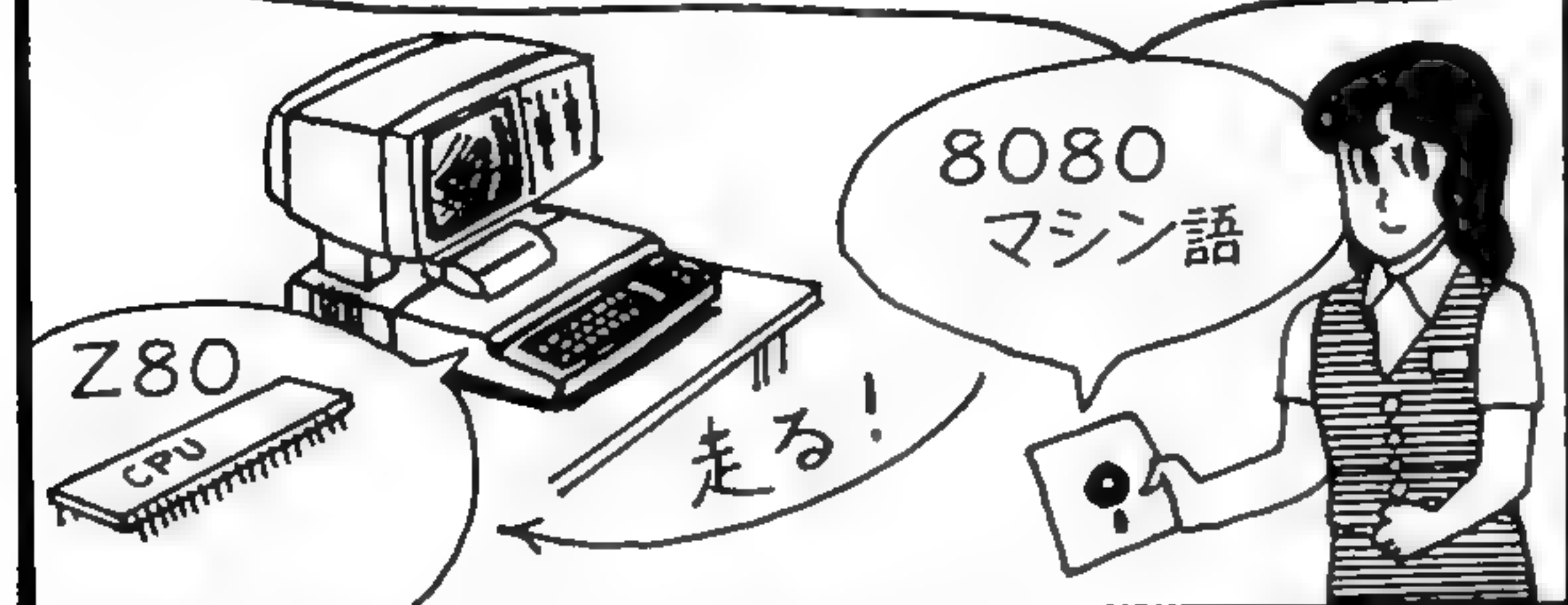
アセンブラやエディタを使うマシン語プログラム開発ツールとしての特色、効用以外の、CP/Mのメリットはなんなのですか。

CP/Mマシン相互間でのソフト・ファイルがコンパチブルで使えるという点が一番でしょうね。





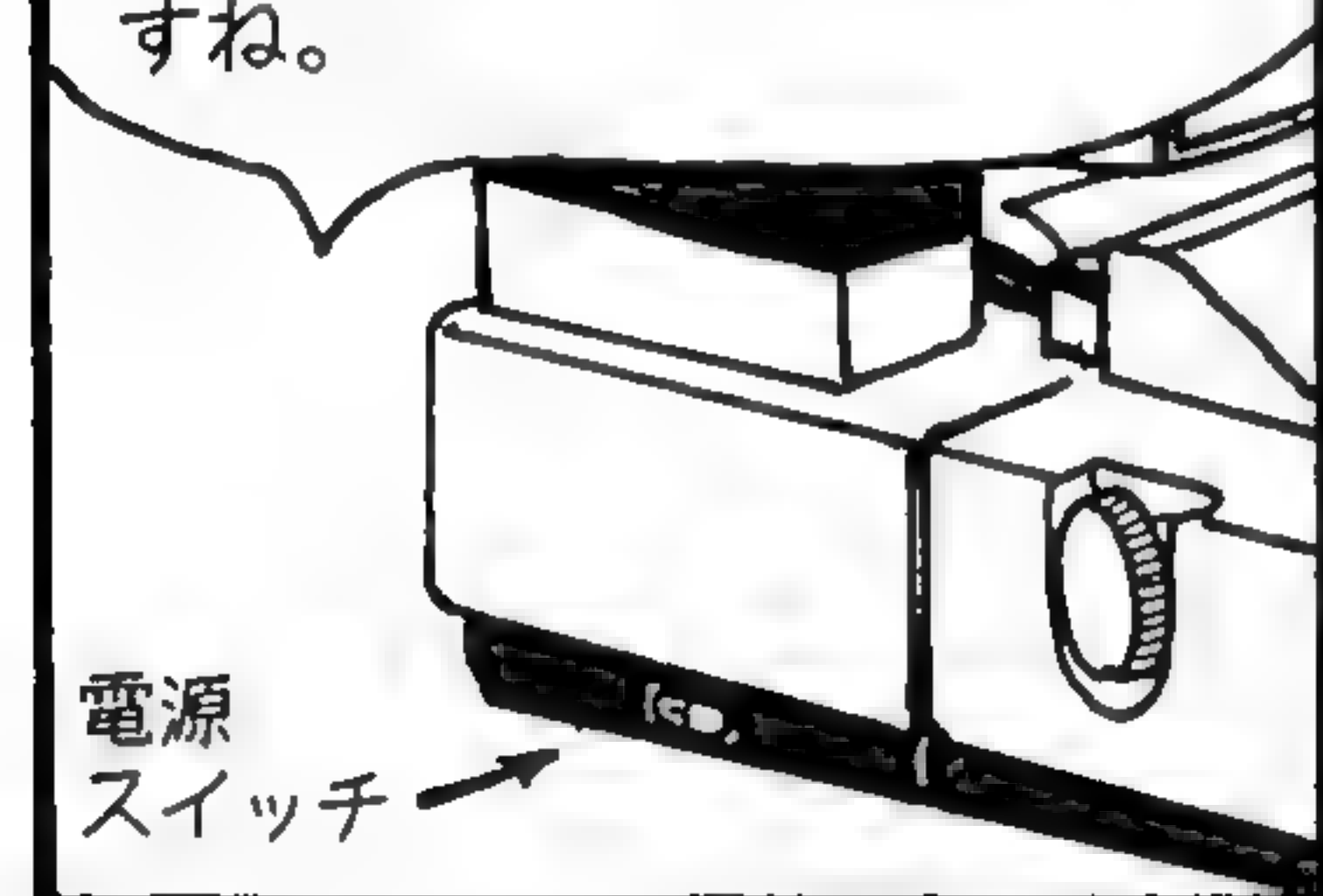
標準システムディスクに入っているのは8080のアセンブラとローダなのです。パソコンのCPUはZ80なので8080のマシン語はそのまま使えるという寸法になっています。



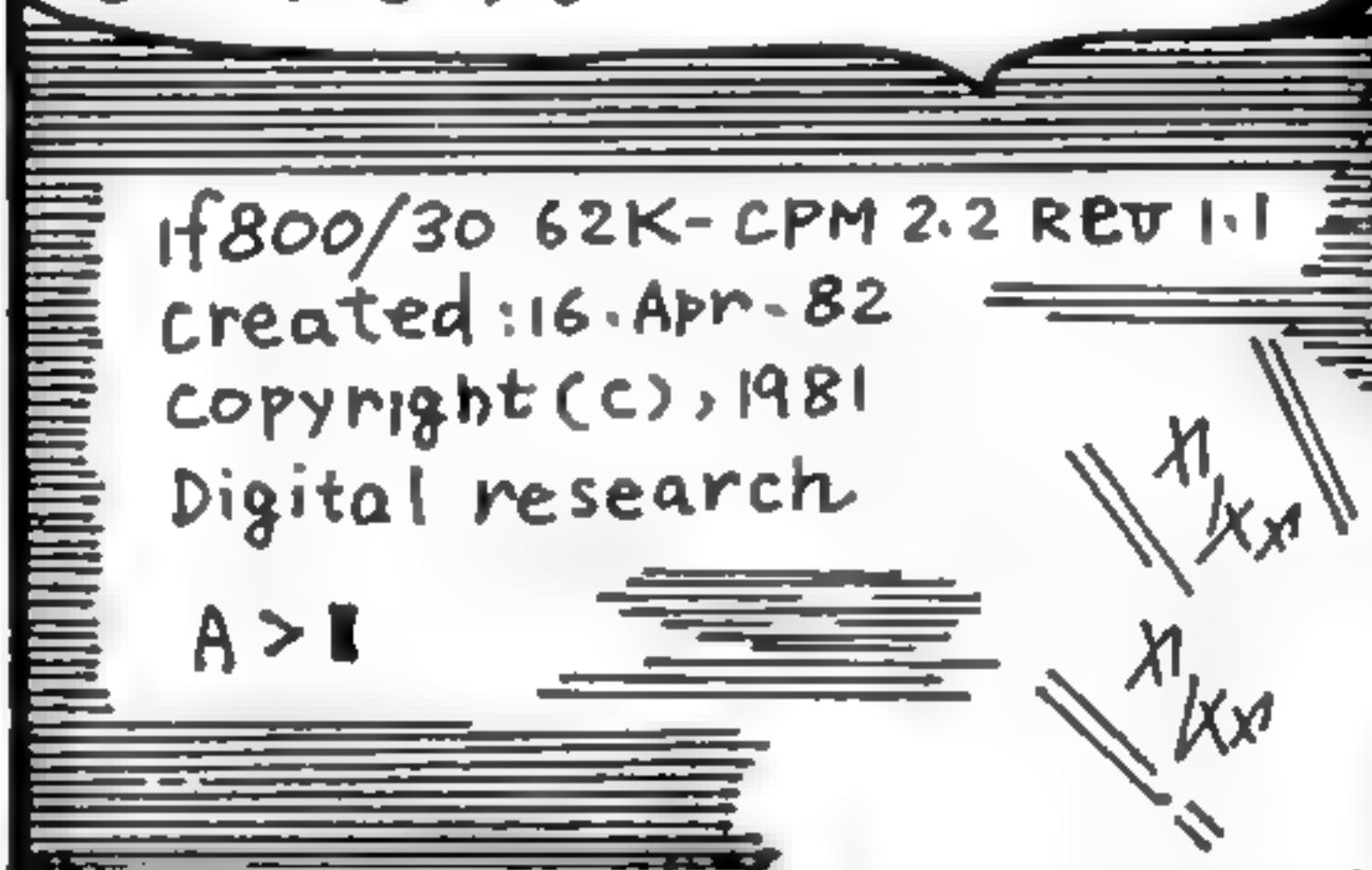
## ②8080アセンブラを走らせよう



このパソコンはオールインワン方式で電源スイッチを1回入れるだけですむのが便利です。



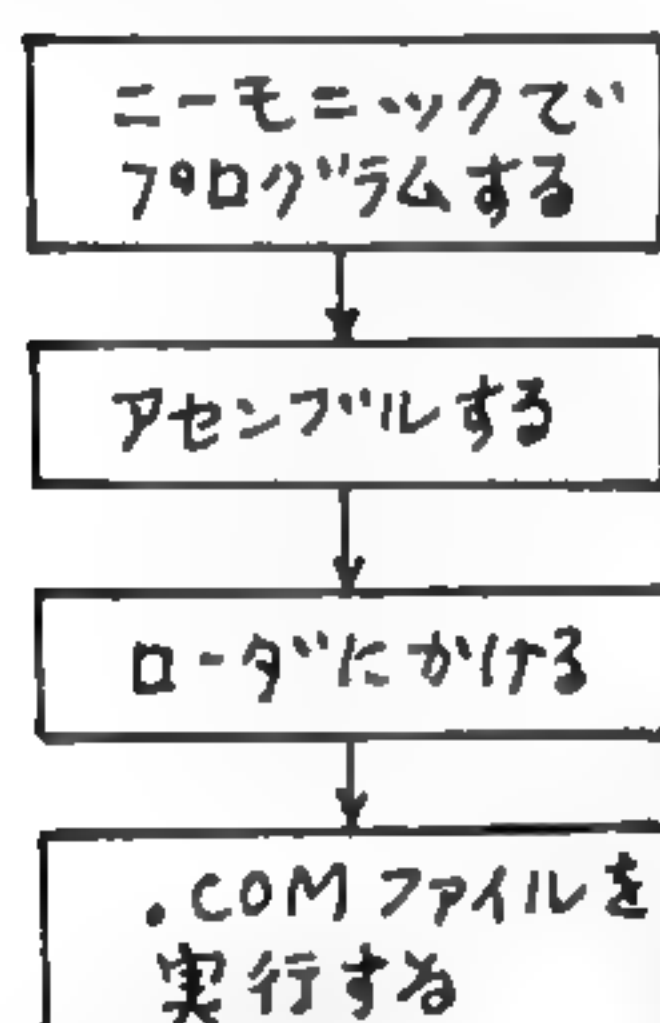
まず、CP/Mのシステムを立ち上げます。A>はCP/Mのプロンプト（コマンド待ち状態）でAはドライブ1を示しています。



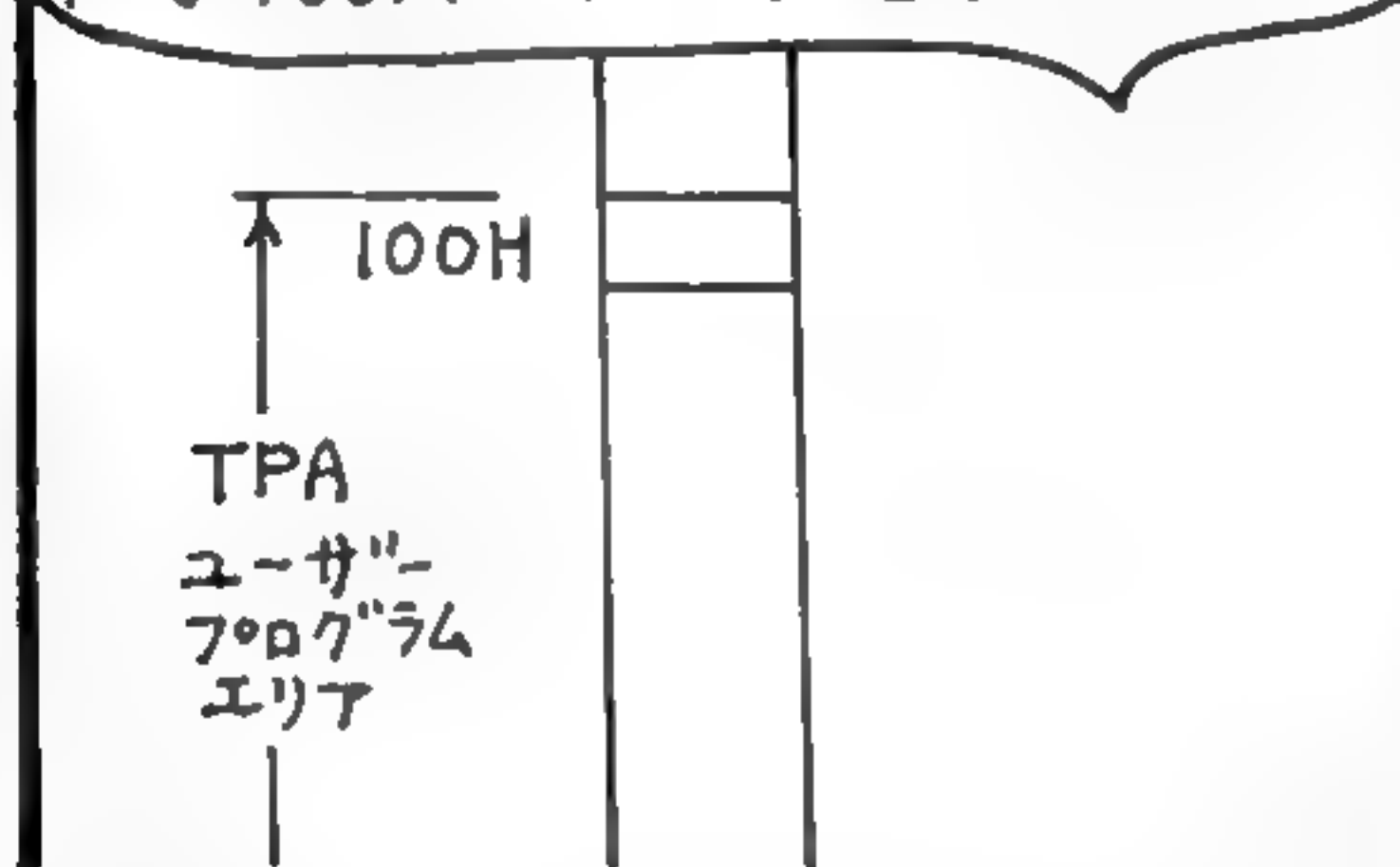
では、このパソコンでニーモニックプログラムをマシン語に変換するという作業をしてみましょう。



今、しようとしているのはこの.COM形式のファイルをつくることなのです。



この.COMは拡張子（エクステンション）でこの場合プログラムがロード後即実行可能なものであることを示しています。COMファイルは第一番目のOPコードが100Hにロードされます。



システムディスクにはCP/Mのほかソフト開発ツールとしてのプログラムも入っていないけません。

FDDUTY.COM  
ED.COM  
ASM.COM  
LOAD.COM  
PIP.COM

このパソコンを買った時ついているものです。

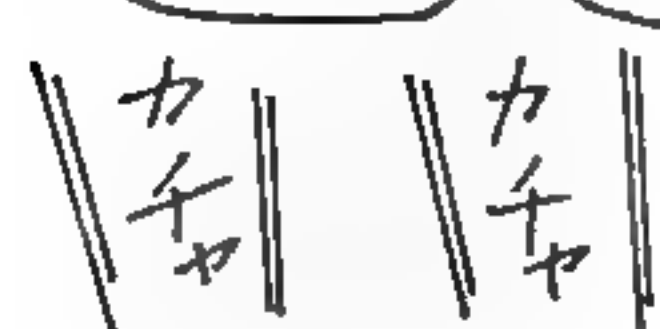


おニューのFDはFDDUTY（フロッピー・ユーティリティー）で初期化してやらねばなりません。

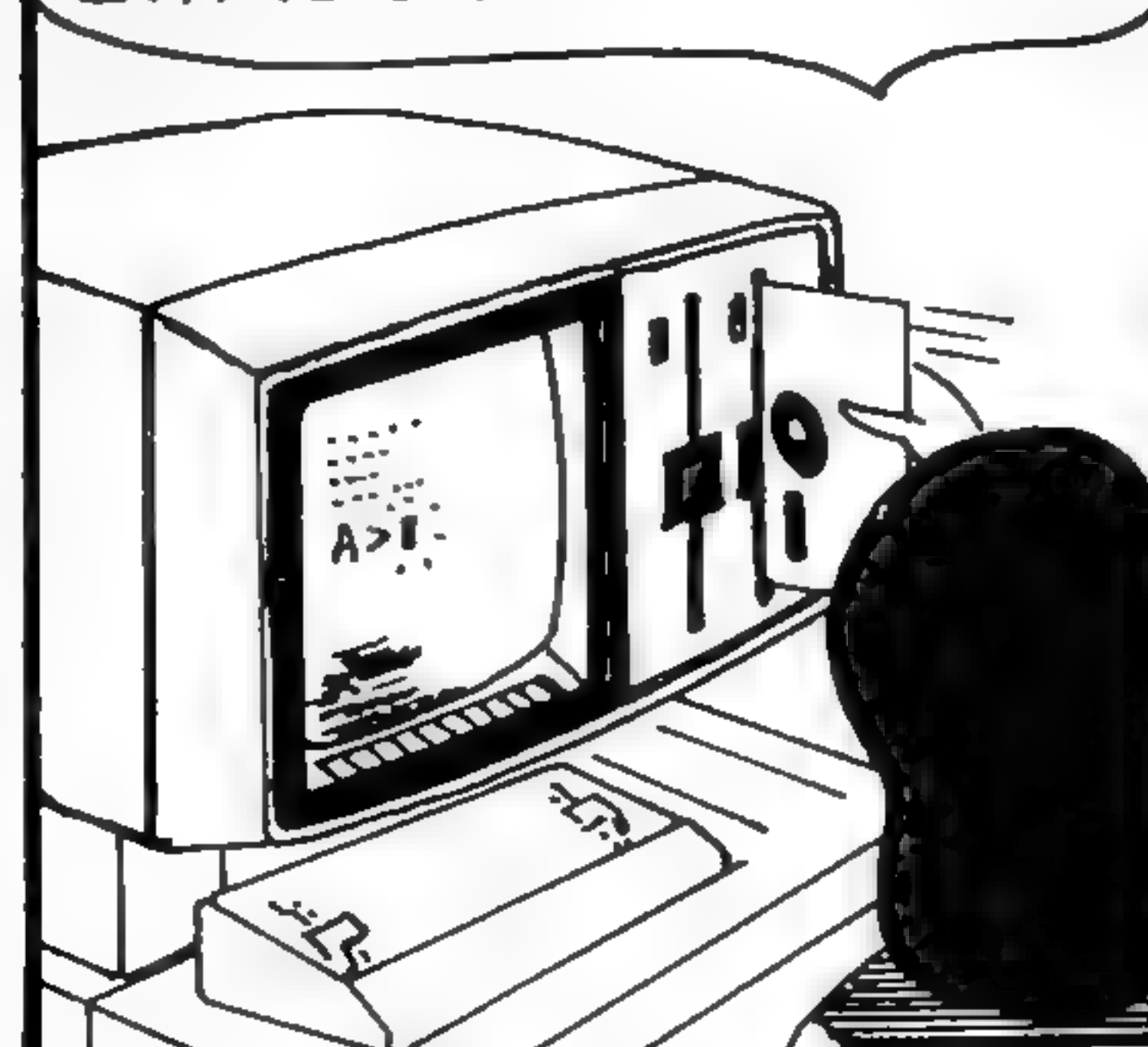
A> FDDUTY

.COMはいりません。

そして RETURN



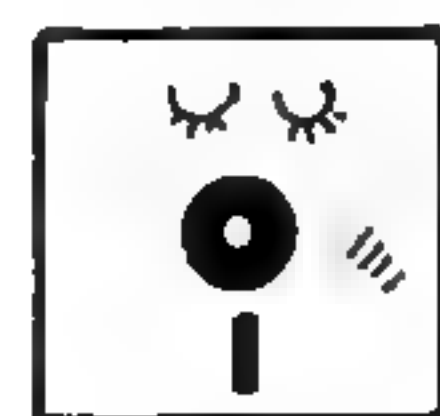
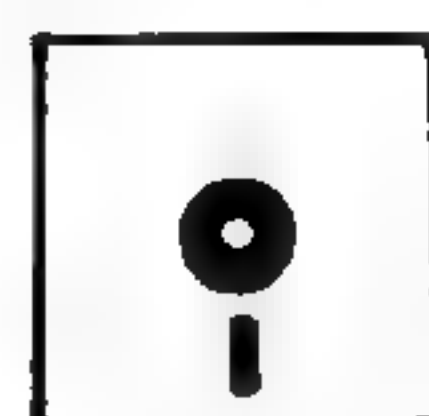
ドライブ2に、おニューのFDを挿入します。



まず、一番にしておかなくてはならないのがこのシステムディスクのバックアップを作っておくことです。マシン語の場合チョンボによっていつプログラムが暴走してフロッピーがオシャカになるかわからないからです。

システムFD

おニューFD





1を入ると、ドライブBにFDDをセットせよ、と表示されます。セットを確認してRETURNキーを押します。

\*\*\* Select function 1  
Mount a disk on drive B:

CP/MではB:はFDDドライブNo.2のことです。

RETURN

このメニューを見て、1~9までの数字をインストすればよいわけです。初期化はメニューから1をインストすればよいことがわかります。

9. Program end  
\*\*\* Select function 1

ここに表示

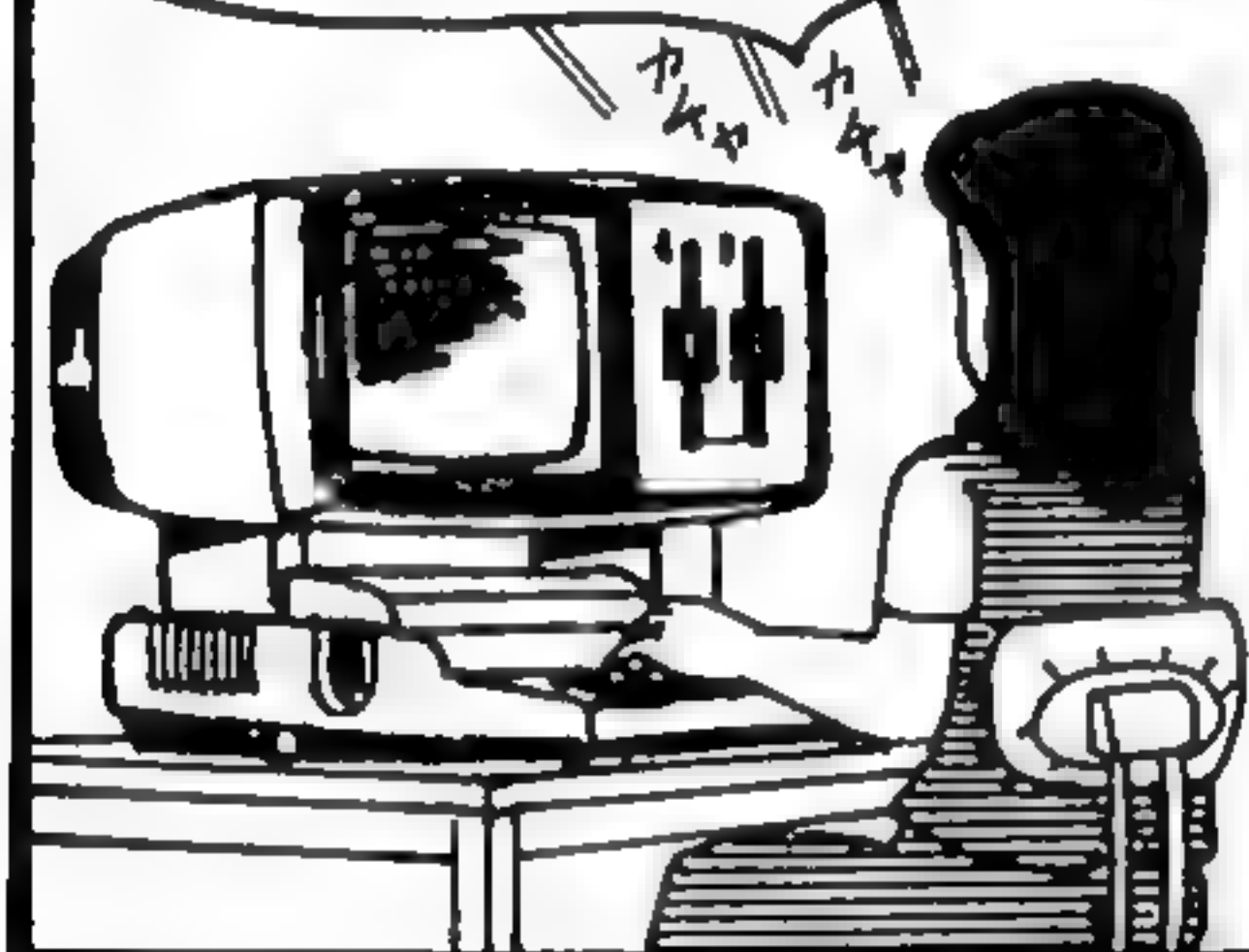
1 RETURN

FDDUTYがロードされると、このようなメッセージが表示されます。

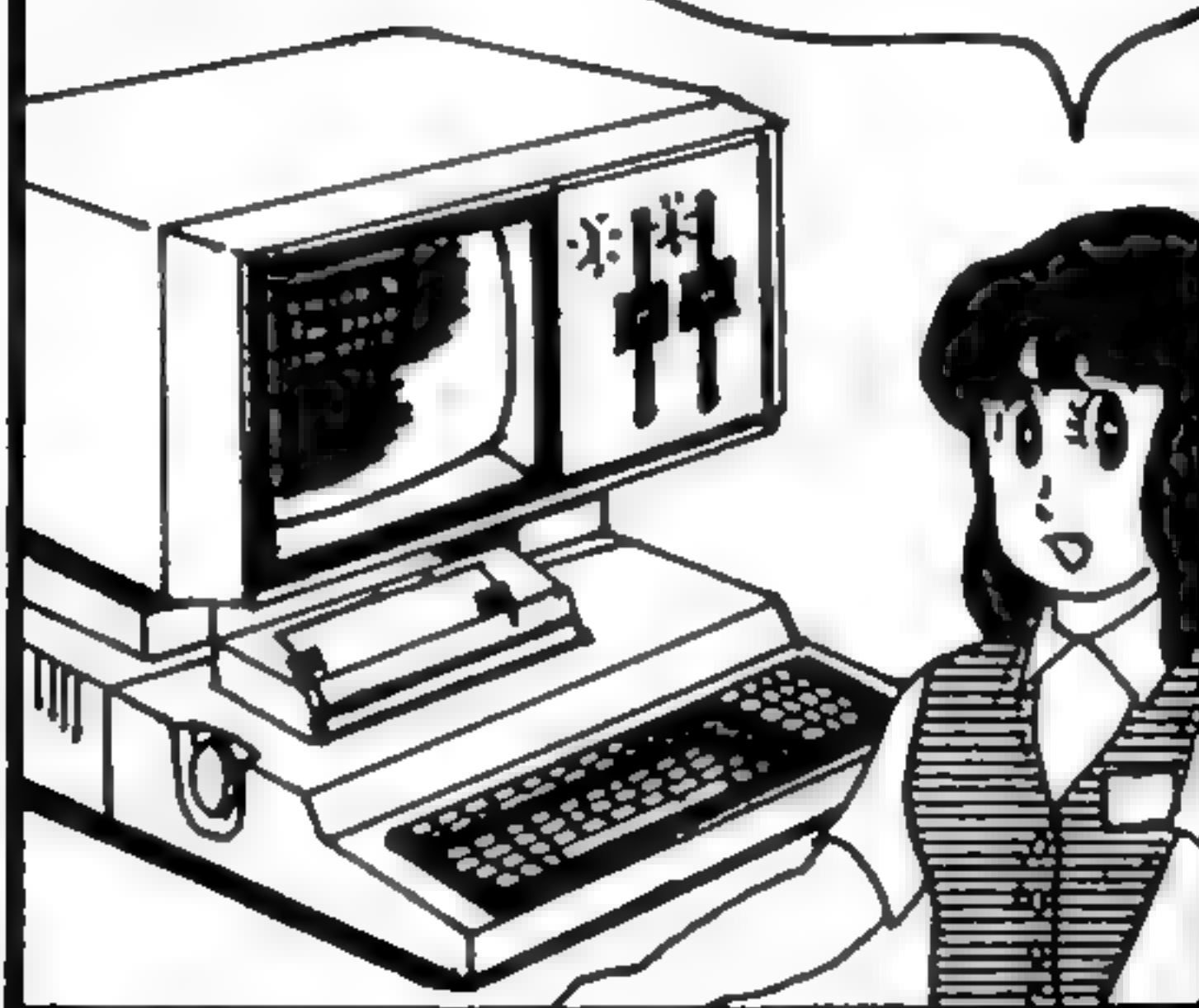
A>FDDUTY  
\*\*\* if 800 model 30 CP/M \*\*\*  
\*\*\* FDD UTILITY PROGRAM Rev 1.0 \*\*\*  
<< program select >>  
1. Address format write  
2. Volume copy

9. Program end  
\*\*\* select function: 1 \*\*\*

トラック76までで処理が終わると、再びメニューのセレクトモードになります。次は2を入れてドライブA:に入っているFDDの内容をそっくりコピーしてしまいます。



この初期化は案外早く終わります。



これでOK。トラックNo.が次々に表示されて処理が進みます。

Mount a disk on drive B:  
Formatting ... type (IS-ID)  
Track - 3

ここが0から76まで次々とカウントアップされる。

DIRノでドライブA:に入っているFDDのファイル名が表示されます。

A: MOVCPM COM: PIP COM:  
A: ED COM: ASM COM:  
A: STAT COM: SYSGEN COM:  
A: BIOS ASM: CBIOS ASM:  
A: FDDUTY COM: KANJI COM:  
A: OBASIC COM:  
:

では、CP/Mのコマンドを少し紹介します。まずFDDの中身を知りたい時のコマンドです。

RETURN  
のキーはノで表現します。



コピーが終わったら、またメニューモードに戻りますので9を入れて作業を終わります。モードはA>です。



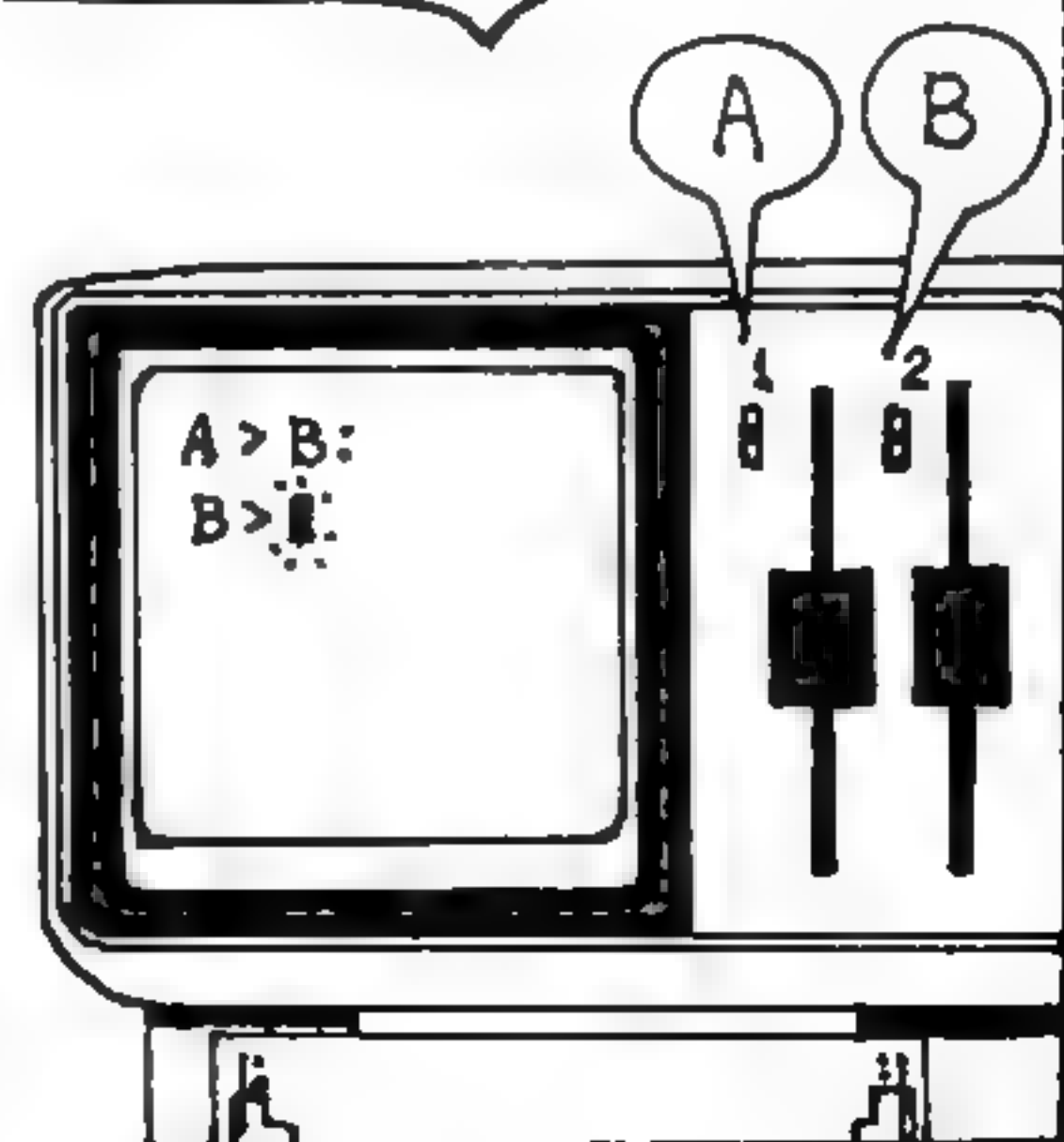
このニーモニックは8088用でした。CP/Mのアセンブラは8080のニーモニックなのです……。

```
ORG 2011
LXI B, 2000
LXI H, 2003
XRA A
MVI E, 03
Loop: LDA X B
ADC M
DAA
STAX B
DCR E
:
```

もう1度A:ノとすると再びAに戻ります。ではCP/Mでソフト開発のマネをしてみたいと思います。テスト用のニーモニックスプログラムをCP/Mの上で走らせてみたいと思います。



ログインディスクA>をB>に変更したい時はB:ノです。Bがログインディスクとなります。





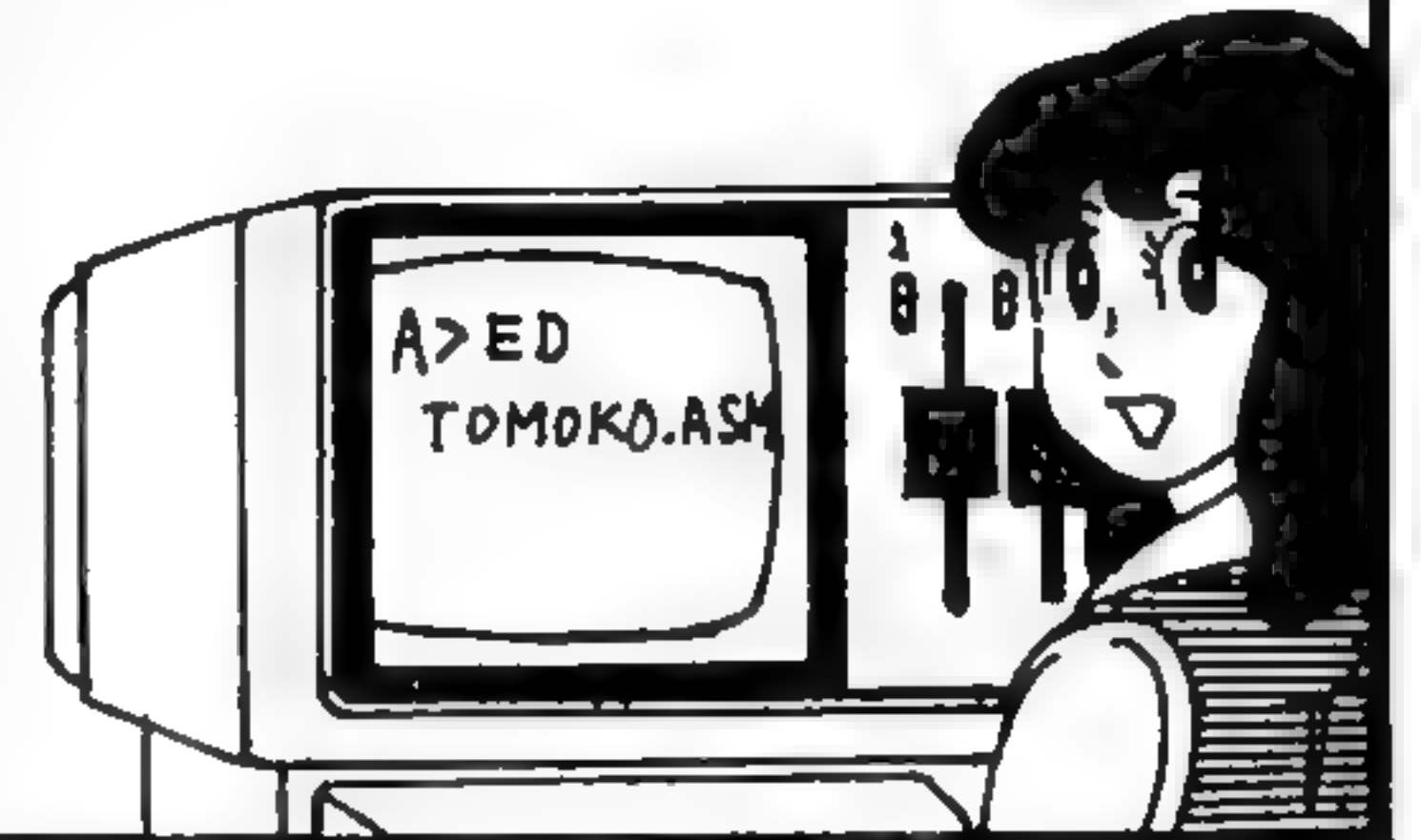
\*はEDのプロンプトです。OLDファイル名で起動した時はAコマンドでファイルからバッファヘデータを移した後に編集しますが、NEWファイルの時はそのままIコマンドでインサートモードに入ります。

```
A>ED TOMOKO.ASM
NEW FILE
:*
:i
1:i
```

これでEDが起動し、メインメモリ内に編集用のバッファができます。ファイル名となるTOMOKO.ASMのASMは必ずつけておく必要があります。NEW FILEと\*が表示しました。

```
A>ED TOMOKO.ASM
NEW FILE
:*
```

手順としてはまずED (エディタ) を起動します。ED.COMファイルがありましたね。このようにキーインをします。ED TOMOKO.ASM



バッファ内にインサートされた二モニックプログラムをFDDにファイルするコマンドはEです。

```
16:END
17:
:*E
```

E

インサートモードの終了はコントロールZです。CTRLキーとZを同時に押します。テキストなどではこの操作を↑ZとかへZとか表現しています。

```
14:JMP LOOP
15:DONE:HLT
16:END
17:
:*
```

次のコマンドを受け付けるプロンプトモードになった。

ここでへZをキーインした。

インサートモードでは文番号が自動的に生成されます。1:、2:、3:など。

```
:*
:i
1:ORG 100H
2:LXI B,200H
3:XRA A
```

ORG 100H  
LXI B,200H  
XRA A

このTOMOKO.ASMをアセンブルするためにASM (アセンブラ) を起動します。ED起動時にはASMをつける必要がありましたが、今度はASMをつけるとエラーです。

```
A>ASM TOMOKO
```

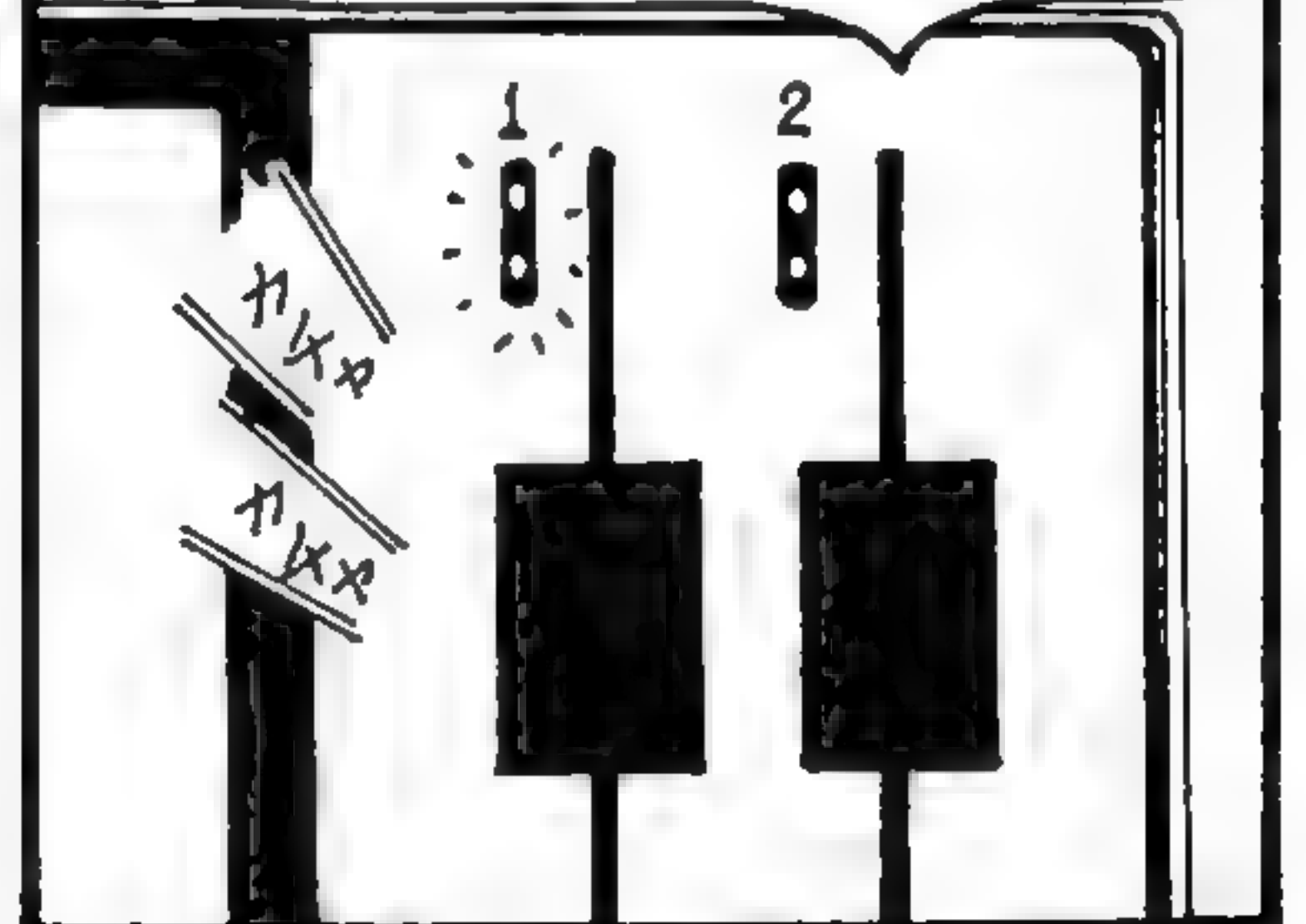
ASM TOMOKO

ASMファイルには今バッファにインサートした新しい内容が入り、BAKファイルには編集前の内容がバックアップとして残されます。

```
16:END
17:
:*E
A>DIR TOMOKO.*
A: TOMOKO BAK:
TOMOKO ASM:
```

```
A>DIR TOMOKO.*
```

コマンドEが実行されるとFDにはED起動時につけたTOMOKO.ASMとTOMOKO.BAKという2つのファイルが生成されます。



次にLOAD (ローダ) を使って、TOMOKO.COMファイルを作ります。ローダはFDからTOMOKO.HEXファイルを読み出し、COMファイルとして生成します。LOAD TOMOKO.HEXとHEXをつけるとエラーになります。

```
A>LOAD TOMOKO
```

LOAD TOMOKO

DIR TOMOKO.\* とキーインするとTOMOKOに関するファイル名だけが表示されます。

```
A: TOMOKO ASM:
TOMOKO BAK:
TOMOKO PRN:
TOMOKO HEX:
```

```
A>DIR
```

TOMOKO で4つのファイルができました。\*はこの場合すべての拡張子という意味です。

FDからASM.COMとTOMOKO.ASMが呼び出されてアセンブルされます。FDDにTOMOKO.PRNとTOMOKO.HEXの2つのファイルを作成してアセンブルが終わります。

```
A>ASM TOMOKO
CP/M ASSEMBLER-VER 2.0
0117
000H USE FACTOR
END OF ASSEMBLY
A>
```



このTOMOKO.COMはCP/M上で即実行できるプログラムです。

A>TOMOKO

TOMOKO✓

TOMOKO.COMファイルができています。DIRでみましょう。

A>DIR TOMOKO.COM

A: TOMOKO.COM

A>!

DIR TOMOKO.COM✓

このように表示されてLOADプログラムの処理が完了します。

A>LOAD TOMOKO

FIRST ADDRESS 0100

LAST ADDRESS 0116

BYTES READ 0017

RECORDS WRITTEN 01

A>!

印字はCP/MではPIP(ピップ)コマンドを使います。PIP LPT:=  
PT:=までが印字のためのコマンドで、その後に印字したいファイル名をキーインすればよいのです。

A>PIP LPT:=TOMOKO.ASM

PIP LPT:=TOMOKO.ASM✓

そこでTOMOKO.ASMを再編集して結果の見えるものにするつもりです。まず今作った各TOMOKOファイルを印字することにします。

でもあれですね。このプログラムは3バイトのたし算なのですが、まだデータが入っていないし、結果を表示するルーチンもないので、このままでは意味がありません。

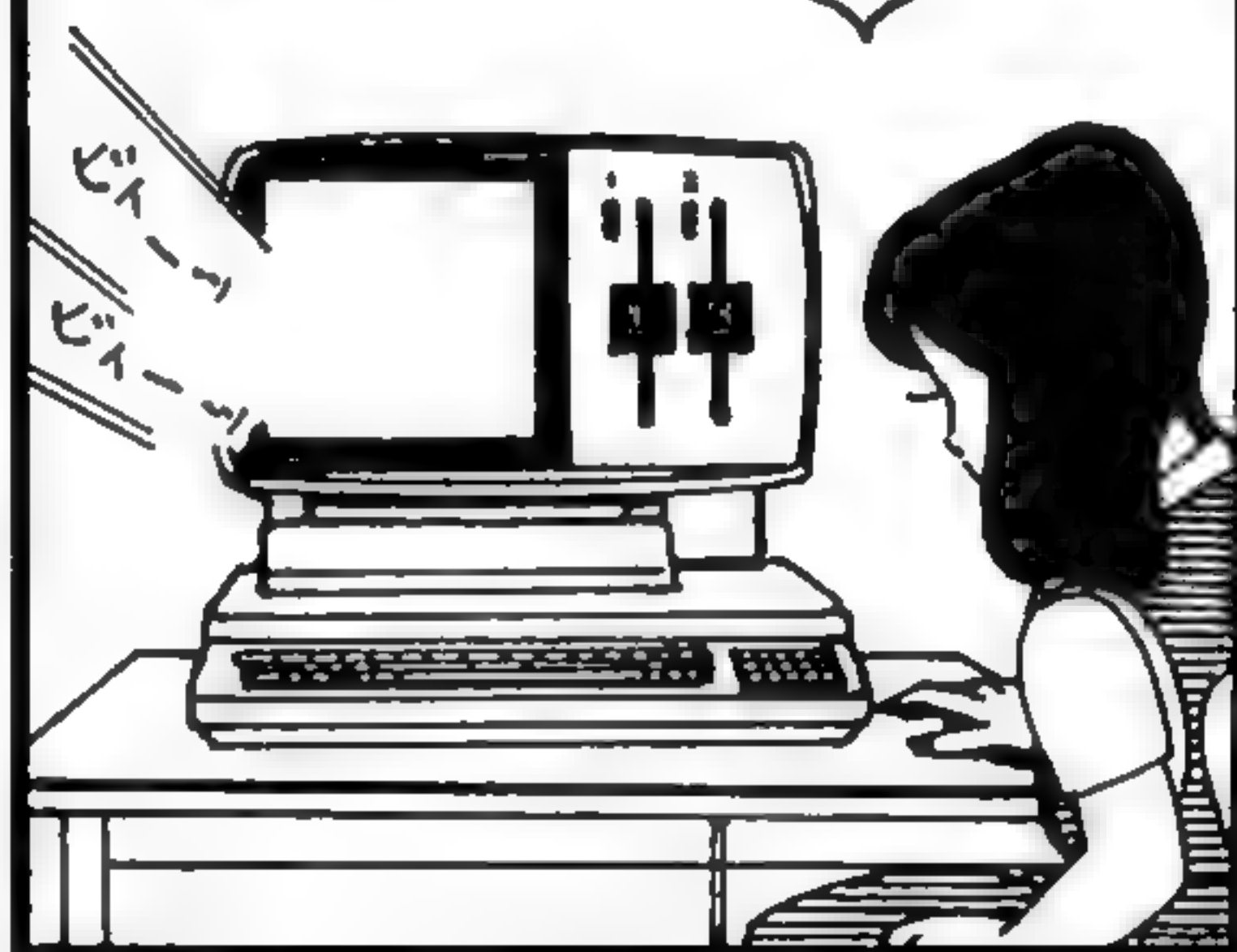


```
ORG 100H
LXI B,2000H
LXI H,2003H
XRA A
MVI E,03H
LOOP:LDAX B
ADC M
DAA
STAX B
DCR E
JZ DONE
INX B
INX H
JMP LOOP
DONE:HLT
END
```

これがTOMOKO.ASMの印字内容です。インサートモードの時に表示されていた行番号は、ありません。

数年前まではこのソースプログラムを紙テープに落としていたのです。

これでTOMOKO.ASMのファイルがロクインディスクA:のFDから読み出されて、印字されます。



0100	ORG 100H
0100 010020	LXI B,2000H
0103 210320	LXI H,2003H
0106 AF	XRA A
0107 1E03	MVI E,03H
0109 0A	LOOP:LDAX B
010A 8E	ADC M
010B 27	DAA
010C 02	STAX B
010D 1D	DCR E
010E CA1601	JZ DONE
0111 03	INX B
0112 23	INX H
0113 C30901	JMP LOOP
0116 76	DONE:HLT
0117	END

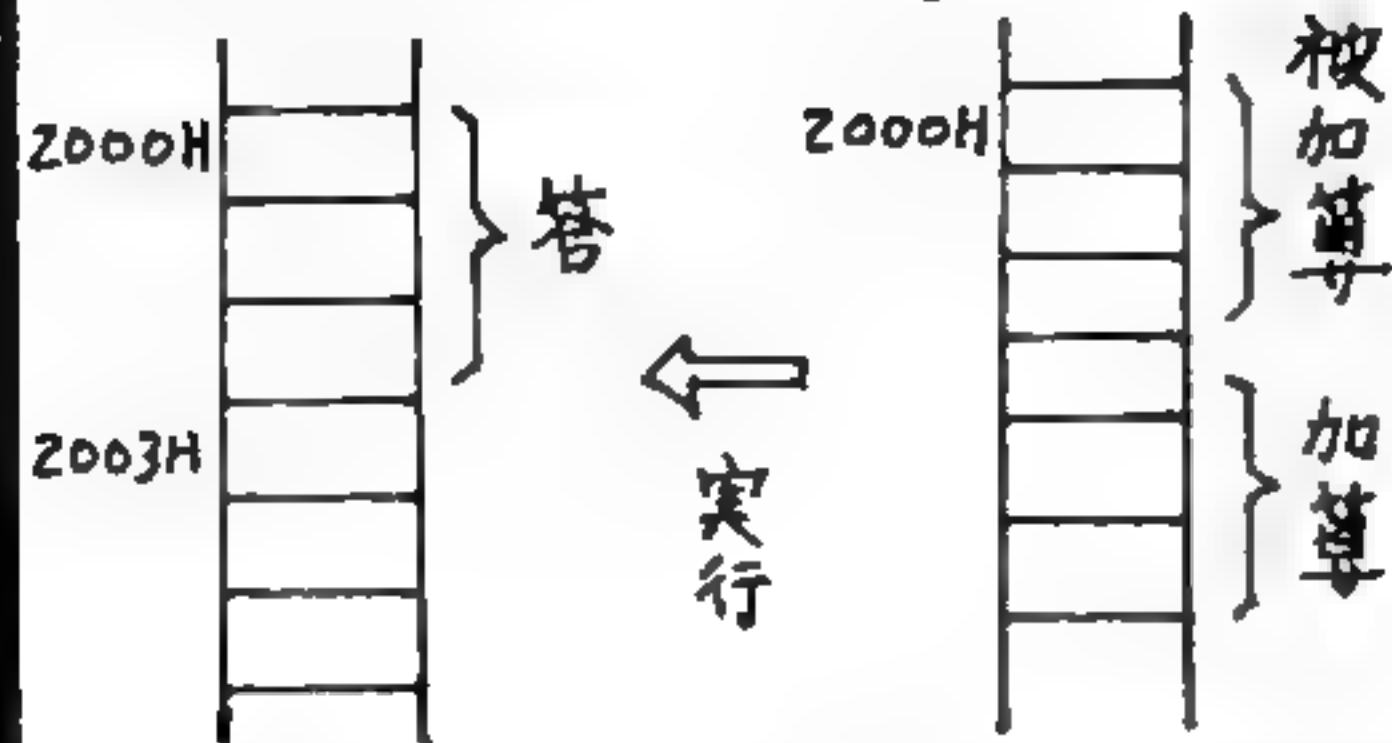
これはハンドアセンブルリストと同じです。テキストでよく見かけるこの書式はPRNファイルだったのです。

同様に PIP LPT:=TOMOKO.PRN✓

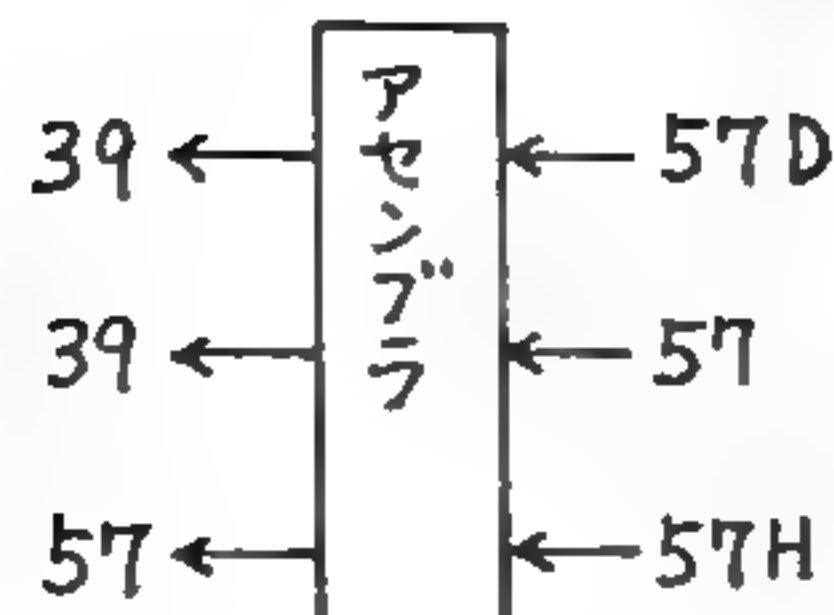




被加算数は2000Hから3バイト加算数は2003Hから3バイトにデータとして入れておき答は2000Hから3バイトに入ります。プログラムが走ったあとは被加算数の代わりに答が入っています。

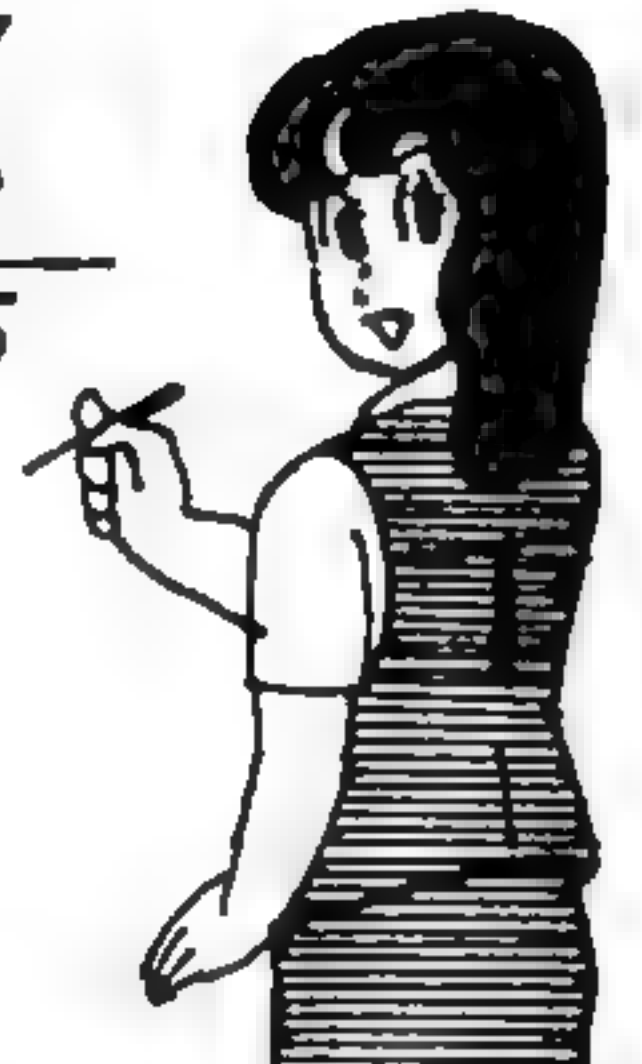


この数値は10進数のつもりですがMVI M, 57Dなどと書くとアセンブルされた時39Hに変換されてしまいます。そこでMVI M, 57Hと書いています。

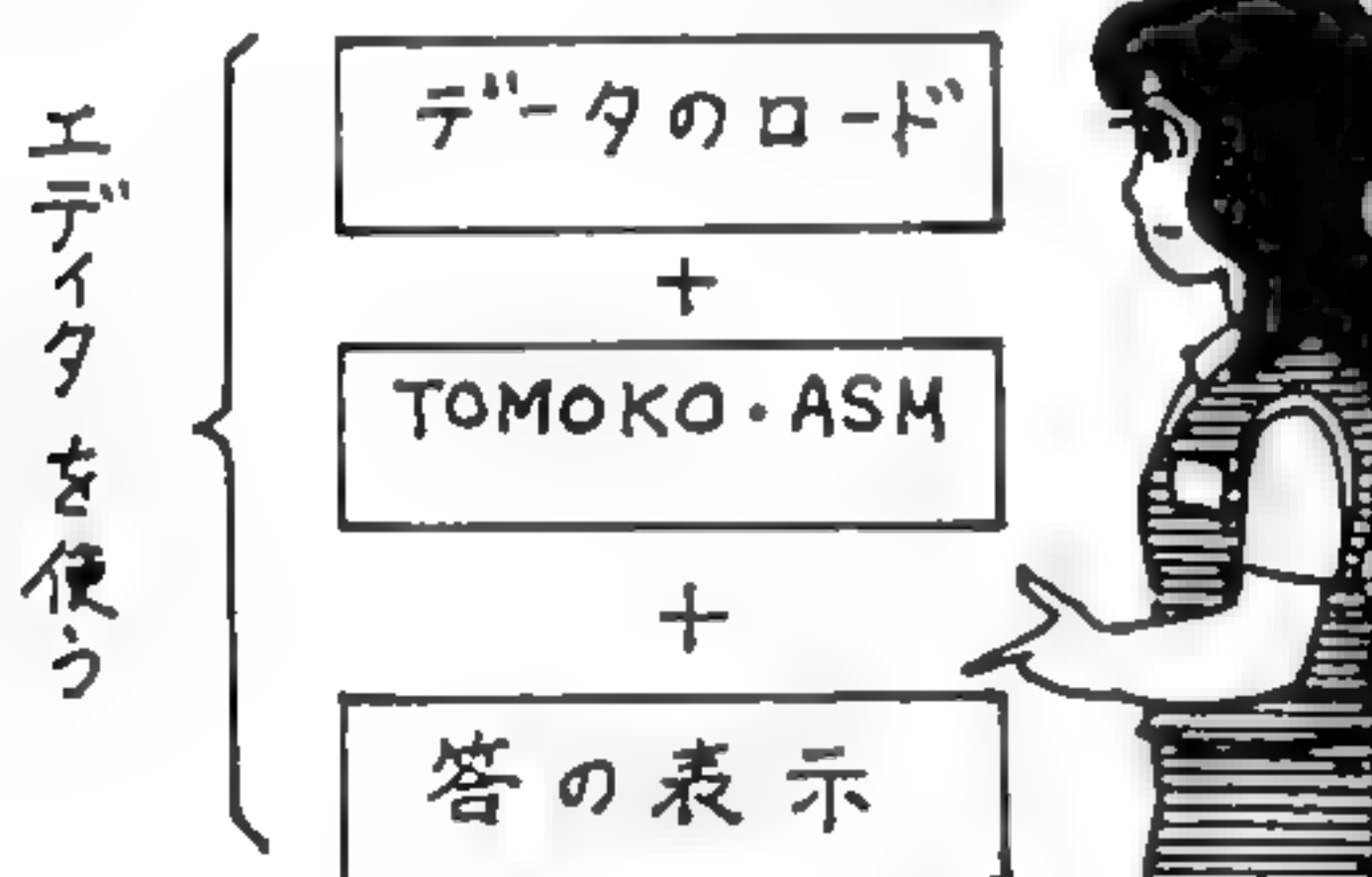


このプログラムは6桁（3バイト）のたし算でした。

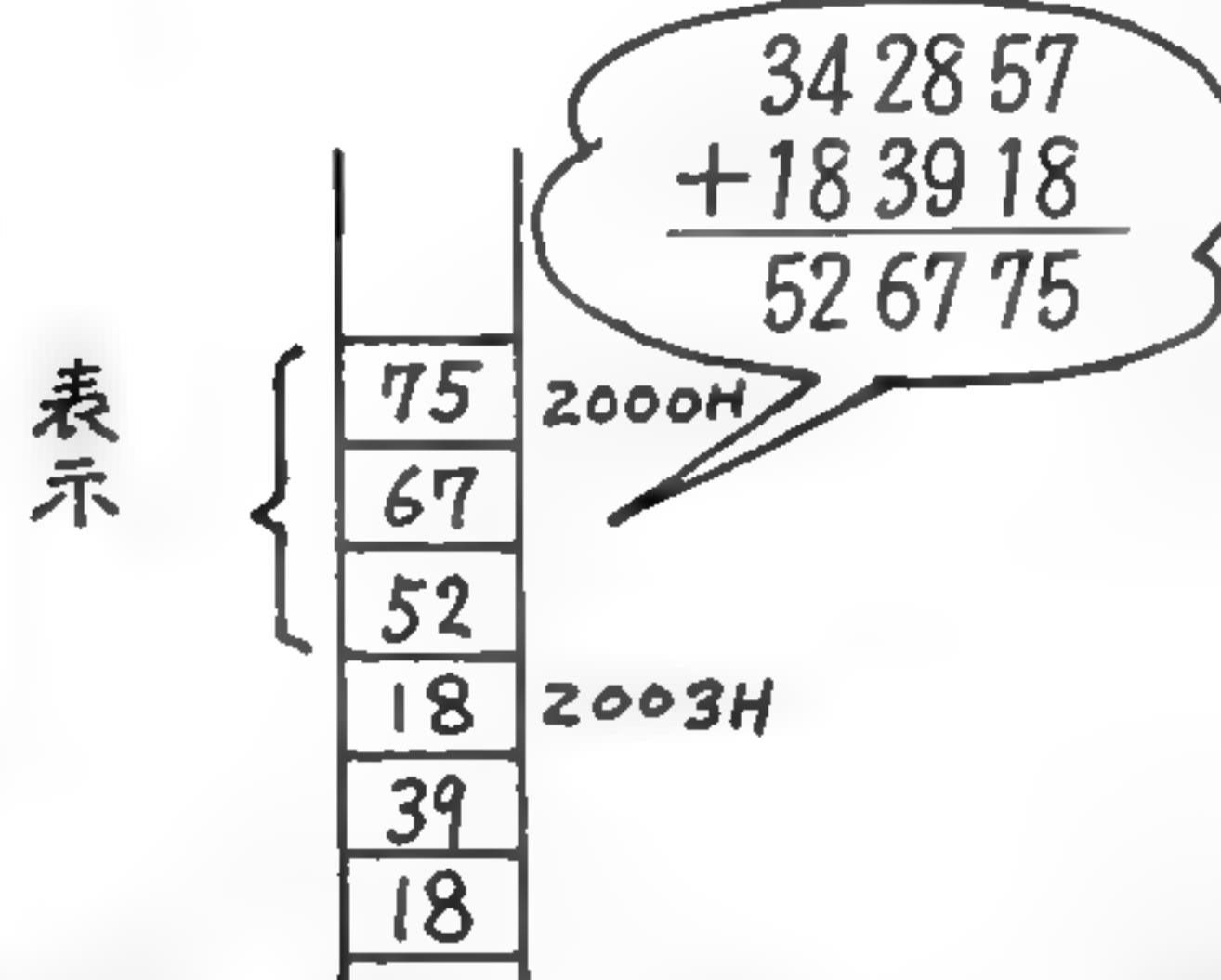
$$\begin{array}{r} 34\ 28\ 57 \\ +\ 18\ 39\ 18 \\ \hline 52\ 67\ 75 \end{array}$$



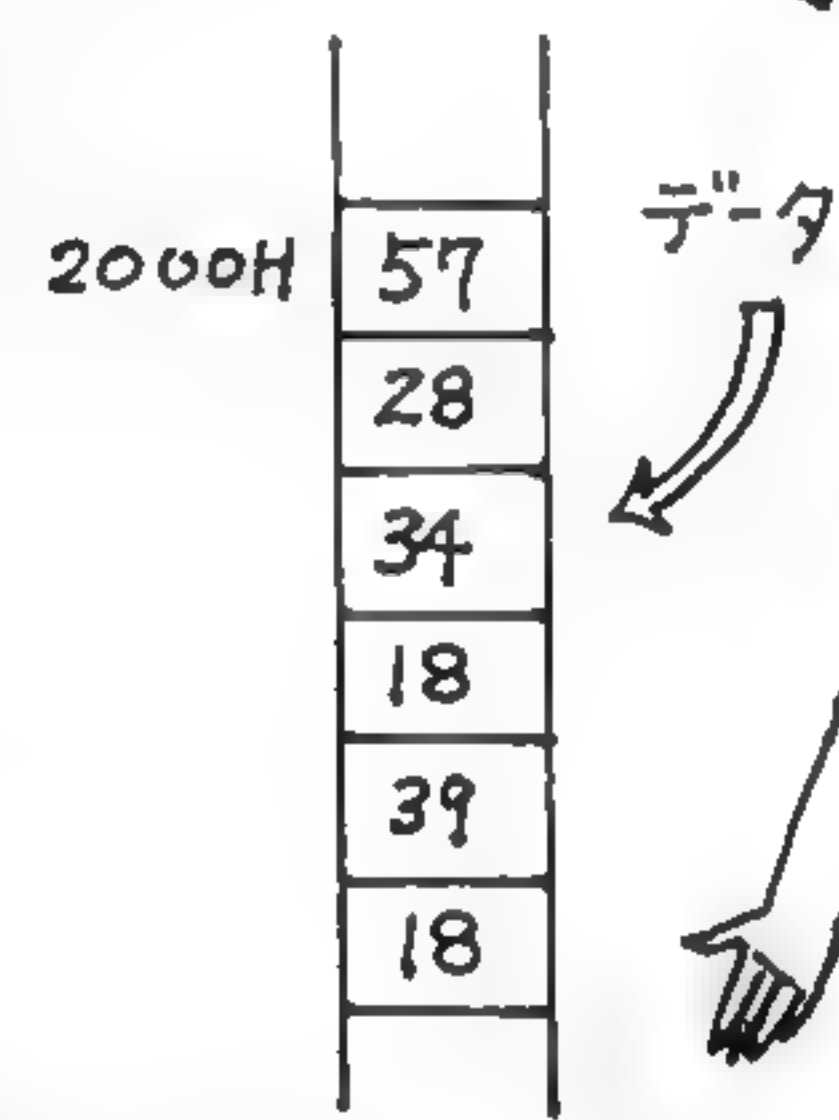
この2つのルーチンを先に作ったTOMOKO.ASMにつけてTAKAKO.ASMというソースファイルを作ってみます。



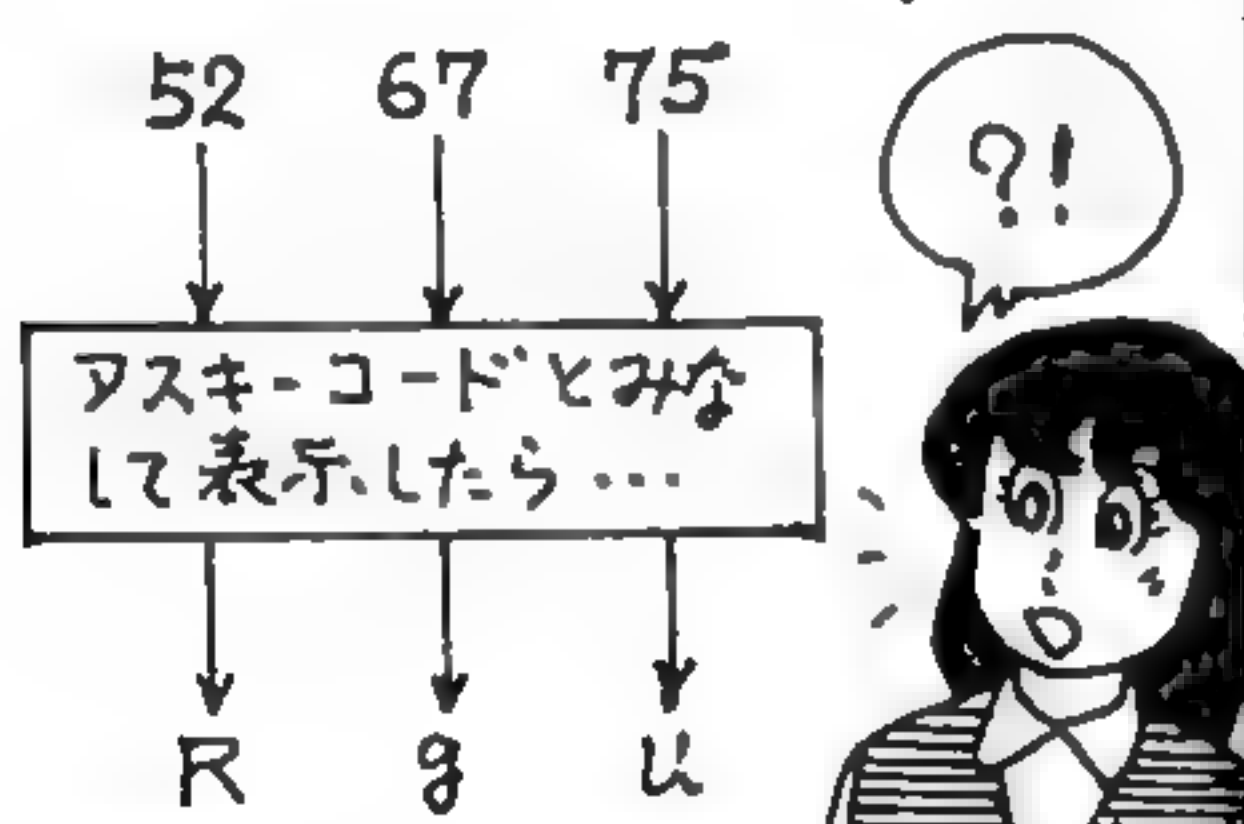
次にTOMOKOの部分を実行した後の2000H~2002Hの内容を表示するルーチンをつけ加えます。



そこでまず2000H~2005Hの6バイトにデータを入れるルーチンが必要になります。



答は52, 67, 75の3バイトですが52をAレジスタに入れてもダメです。52はアスキーコードではRです。つまりこの場合R・g・uと表示されてしまうのです。



表示はCP/Mのシステムコールと呼ばれるサブルーチンを使います。1バイトのアスキーコードをキャラクタで表示します。

```
MVI C, 2
MOV E, A
CALL 5
```

これでAの内容が表示されました。



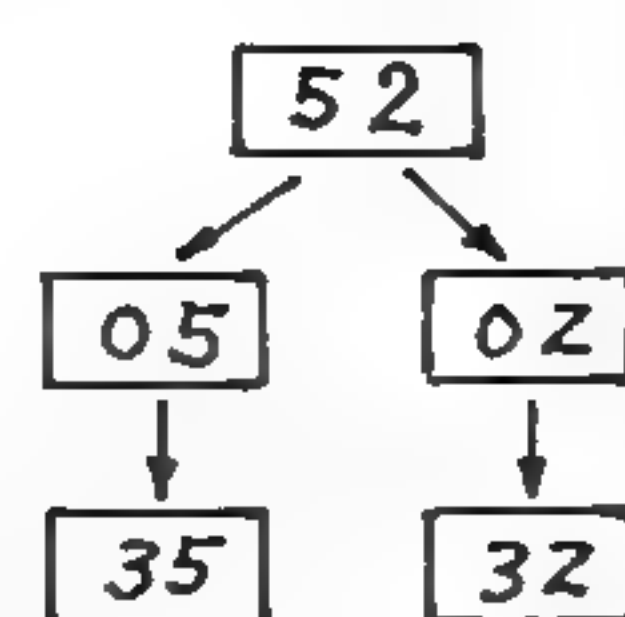
まず定数をメモリに書き込むルーチンです。(データのロード)

```
ORG 100H
LXI H, 2000H
MVI M, 57H
INX H
MVI M, 28H
INX H
MVI M, 34H
INX H
MVI M, 18H
INX H
MVI M, 39H
INX H
MVI M, 18H
```

1バイトで表現される52を5と2に分解してアスキーコードの35と32にします。アスキーでファイルに書き込む時もこのように多くのバイトが必要になると思われれます。

内部表現					
52	67	75			
↓ アスキー表現					
35	32	36	37	37	35

そこで5・2・6・7・7・5とそれぞれのキャラクタを個別にアスキーコードで扱う必要があります。この場合は全部数字ですから頭に3をつけて35, 32, 36, 37, 37, 35とすればアスキーコードになります。フラグ(スペース)コードは20です。



あまりスマートではありませんが、これでもいいはずです。

インテル8080の命令表を参照してください。

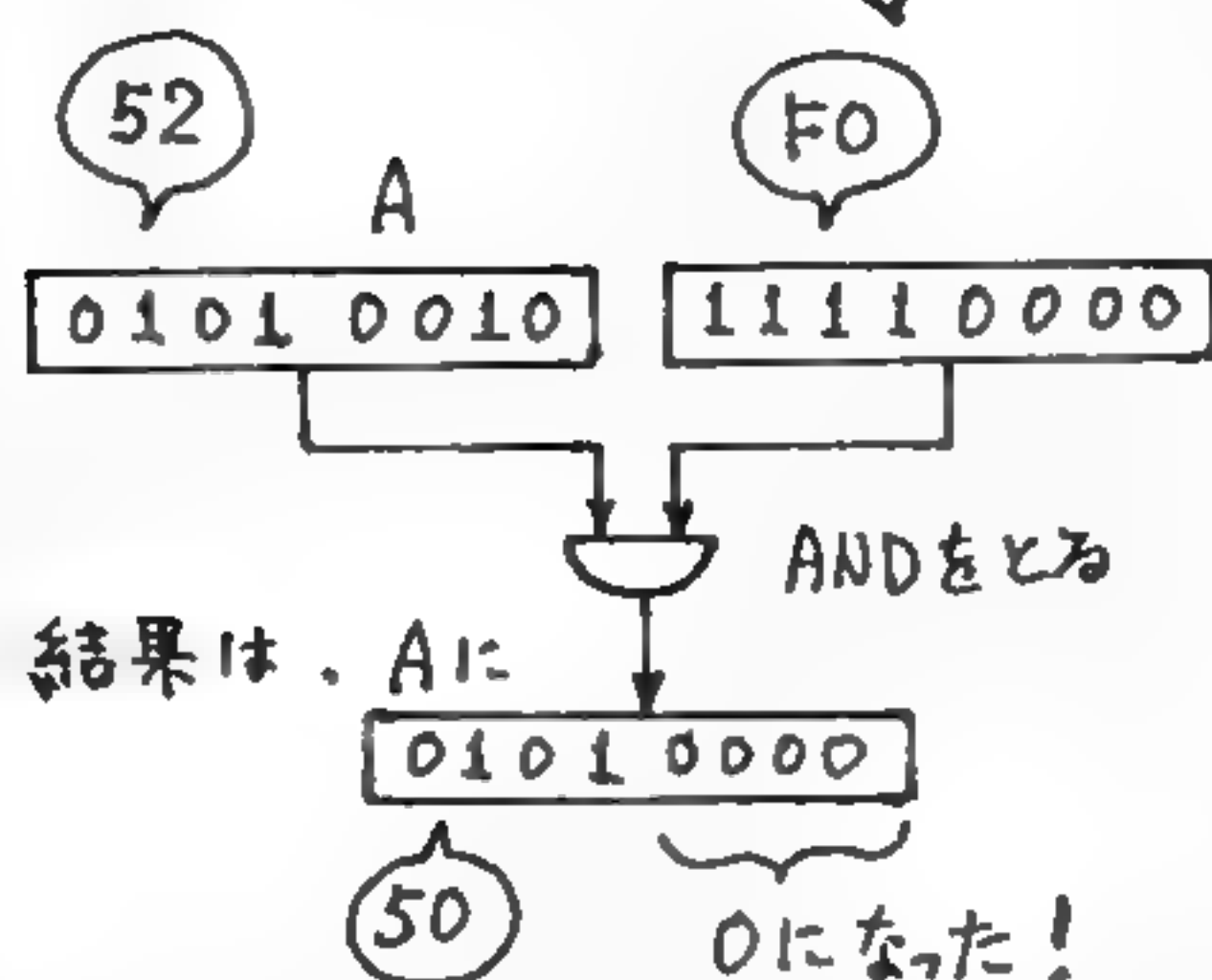




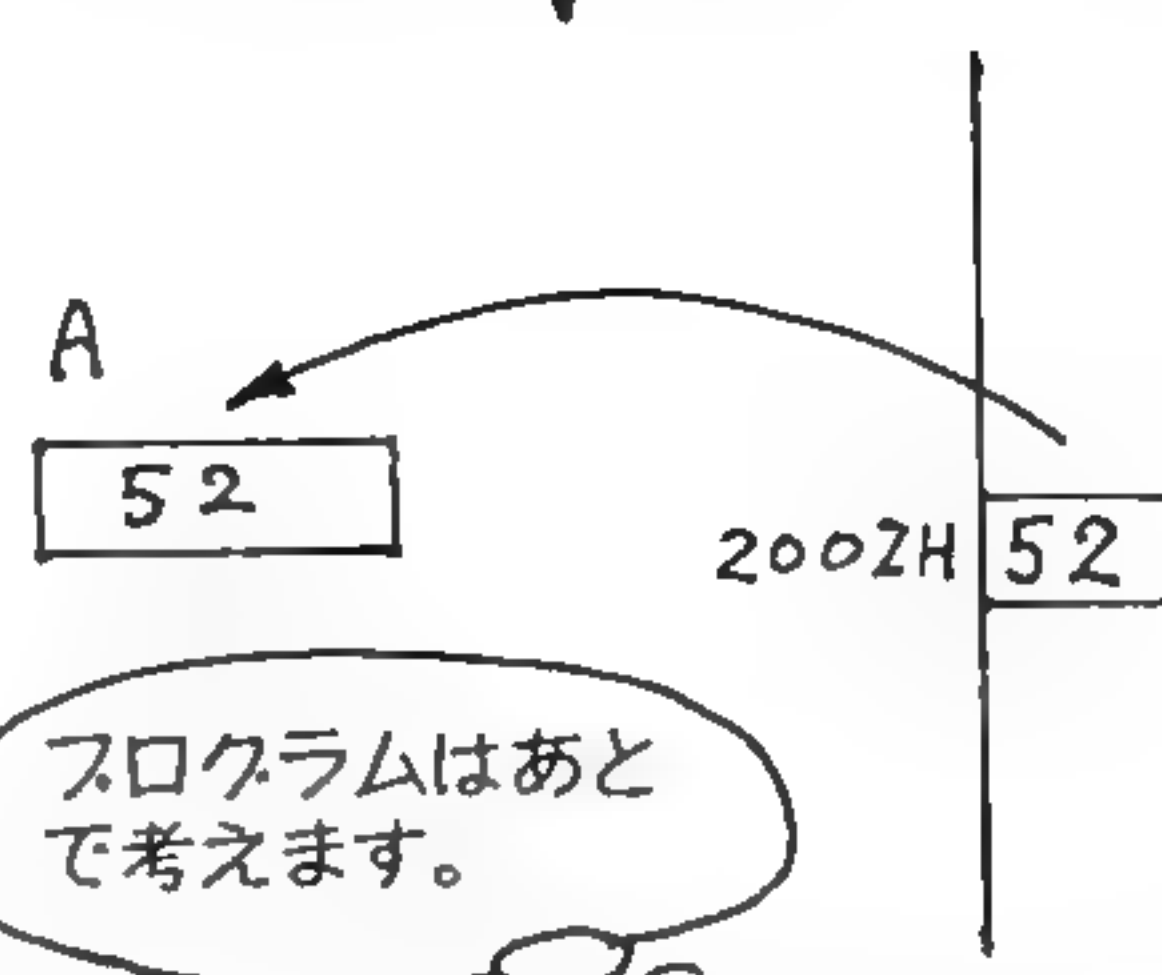
この状態でAレジスタの内容は52から50に変わっています。ここで、さらにAレジスタを4ビット右に回転してやると05になります。

```
0101 0000 RRC (回転)
0010 1000 RRC
0001 0100 RRC
0000 1010 RRC
0000 0101
```

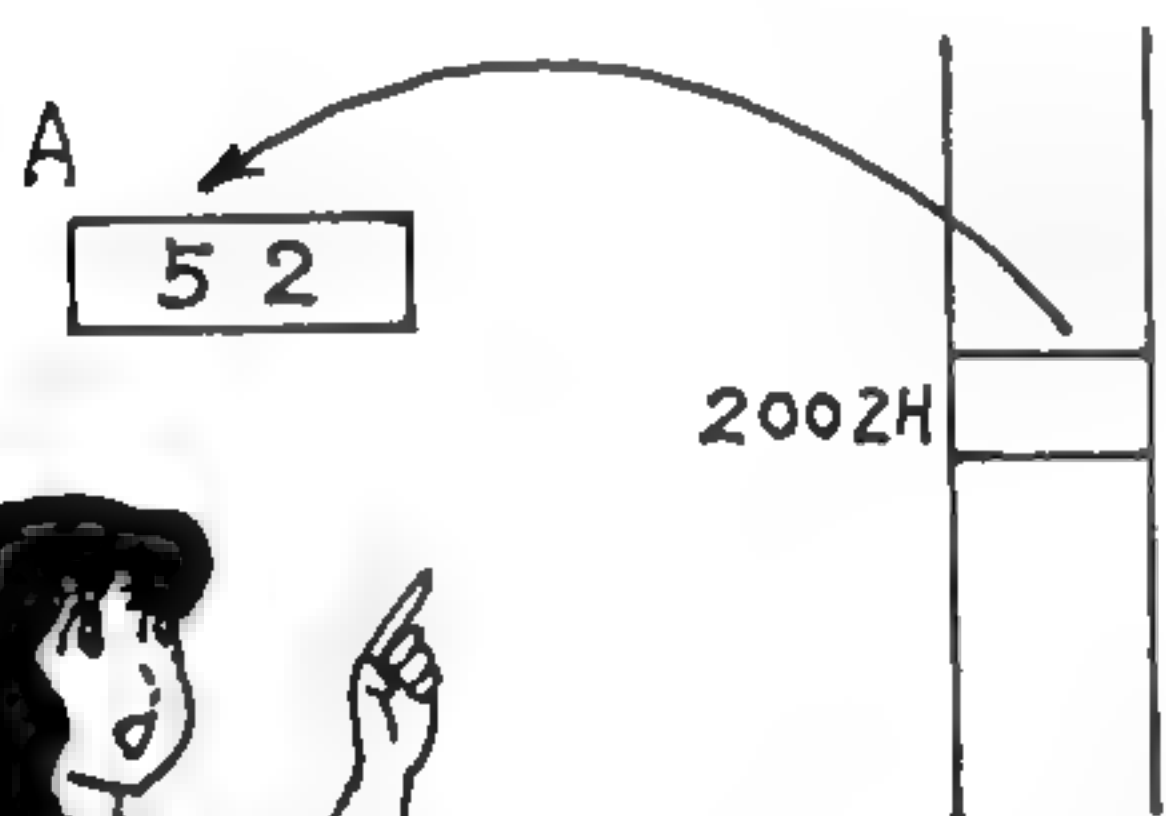
次にAレジスタの下4ビットをマスクするためにF0とのANDをとります。



具体的にはAレジスタを使ってビット操作します。まず、答の先頭の2002Hの内容(52)をAレジスタに入れます。



下桁の2はもう少し簡単です。もう一度2002Hから52をAレジスタに入れます。



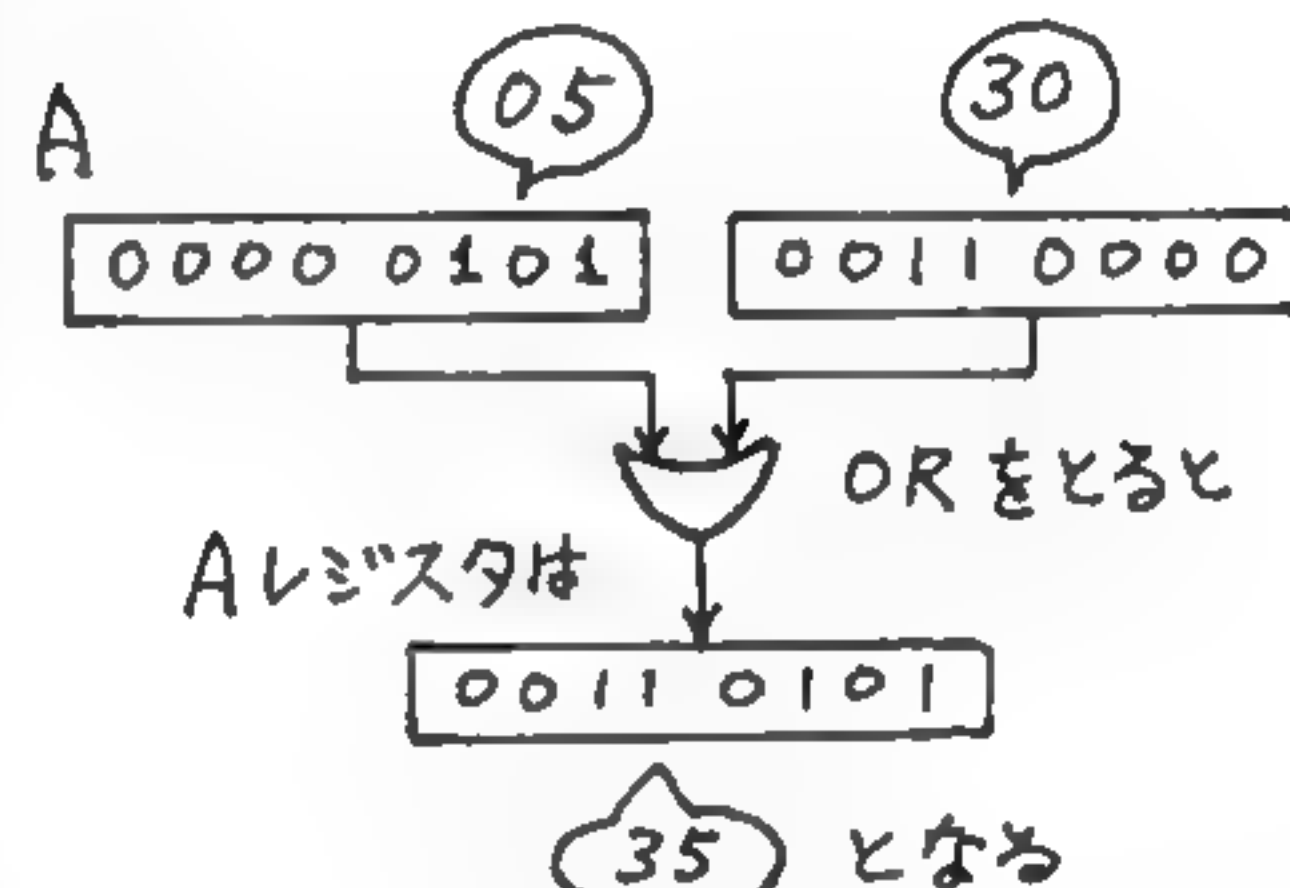
このAレジスタの内容35Hを使ってシステムコール(ファンクション2)をすれば'5'が表示されるわけです。

35 表示用コード

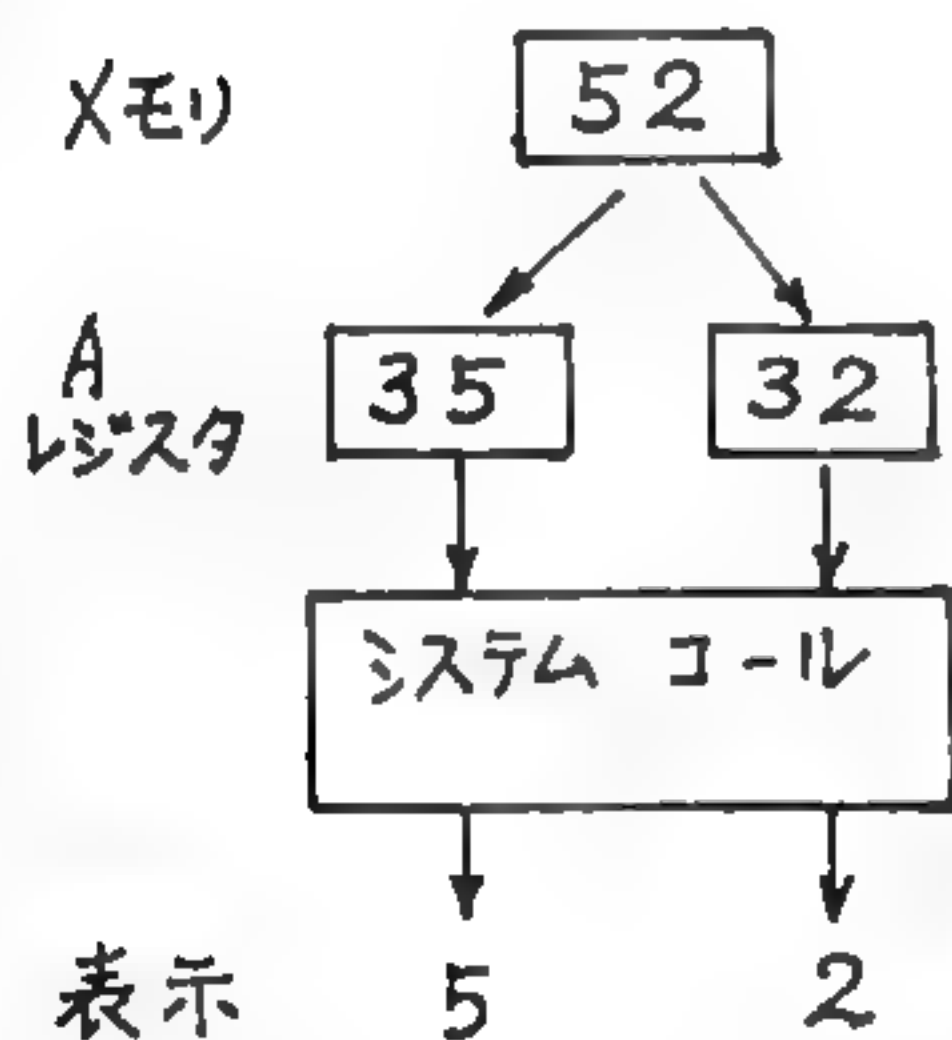
```
MVI C, 2
MOV E, A 35
CALL 5
```

このシステムコールはCP/Mに用意されたサブルーチンです。

この05と30とのOR(オア)を取ってやればAレジスタの内容は35となります。これでキャラクタ'5'のアスキーコード35ができました。



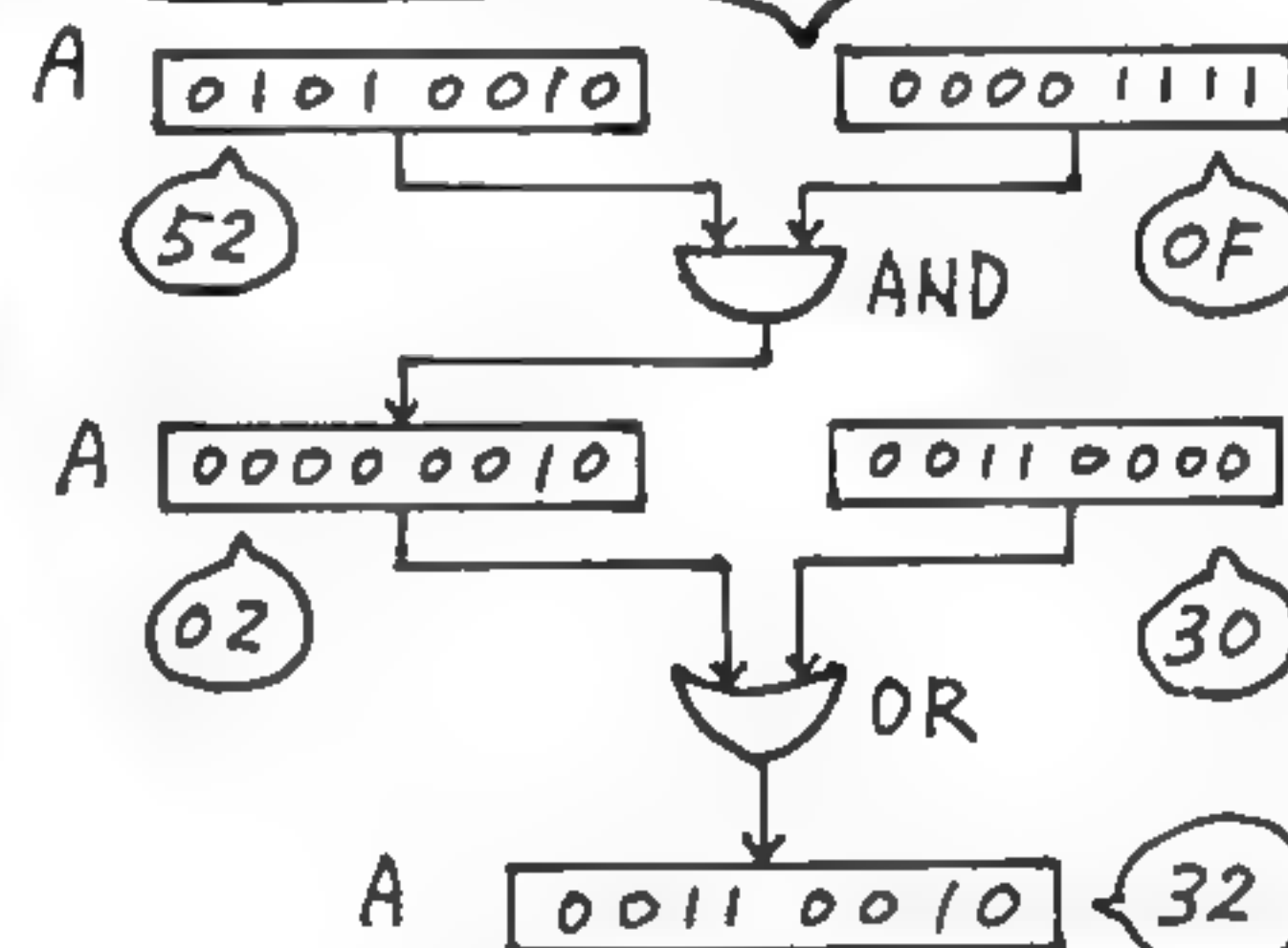
これで1バイト分の表示ができました。



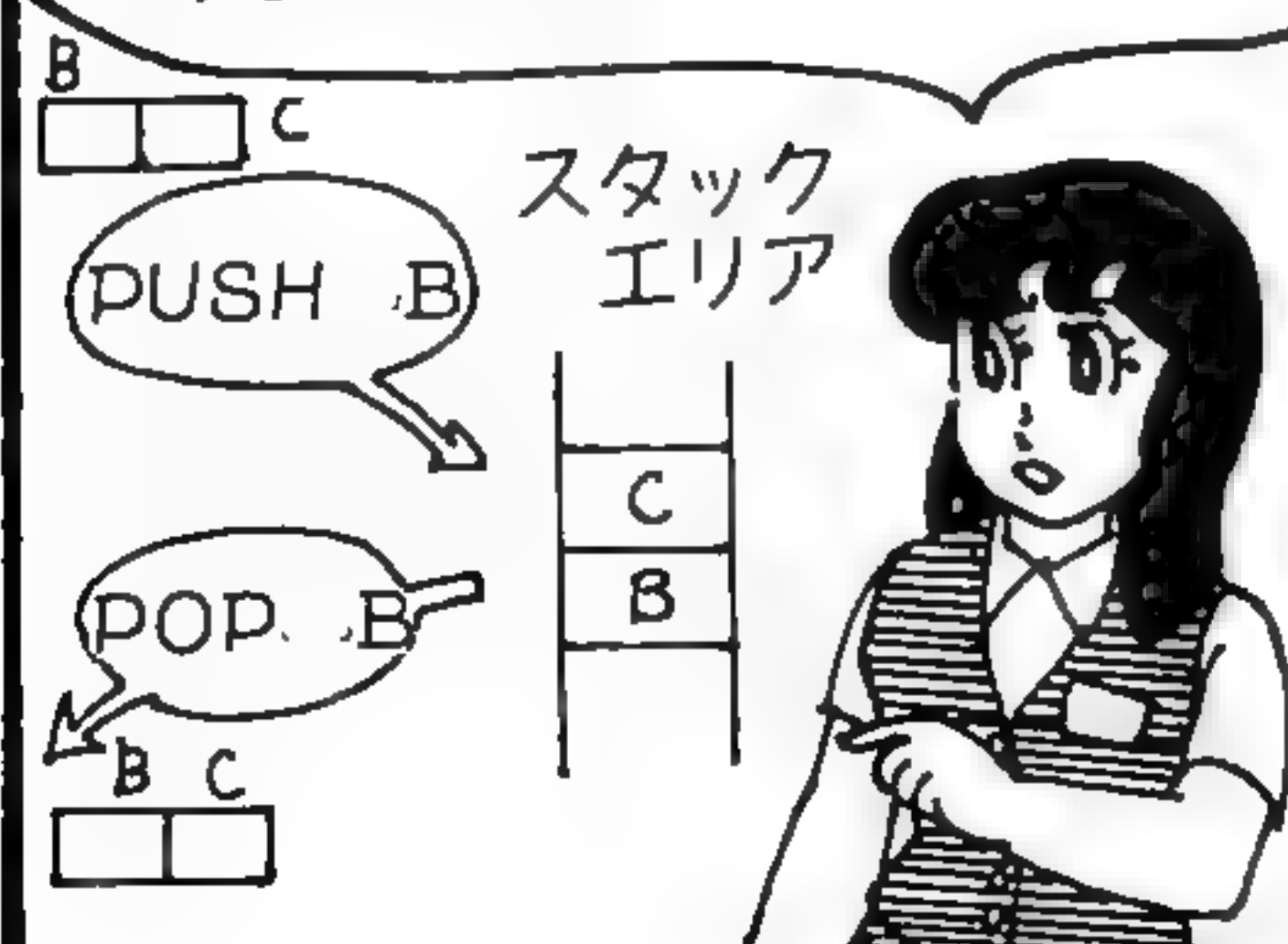
32HとなったAレジスタの内容をEレジスタに入れ、システムコールすれば'2'が表示されます。

```
MVI C, 2
MOV E, A 32
CALL 5
```

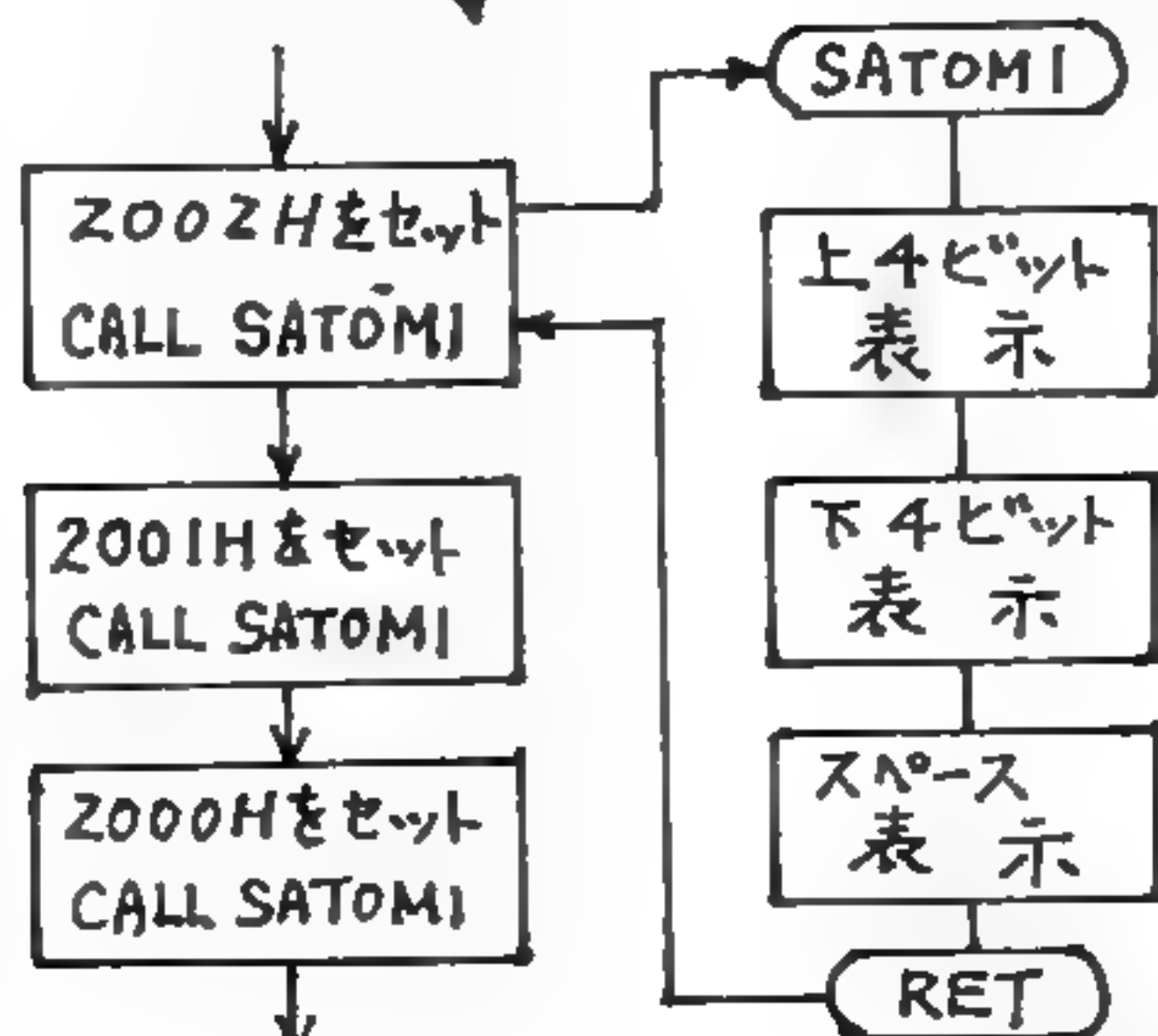
今度は上4ビットを0FとのANDを取ってマスクし、これと30とのORを取ってやれば、'2'のアスキーコード32が得られます。



表示部のルーチンができました。アドレス指定にBCレジスタペアを使いシステムコールでCレジスタを使うため、PUSH、POP命令でデータを守っています。

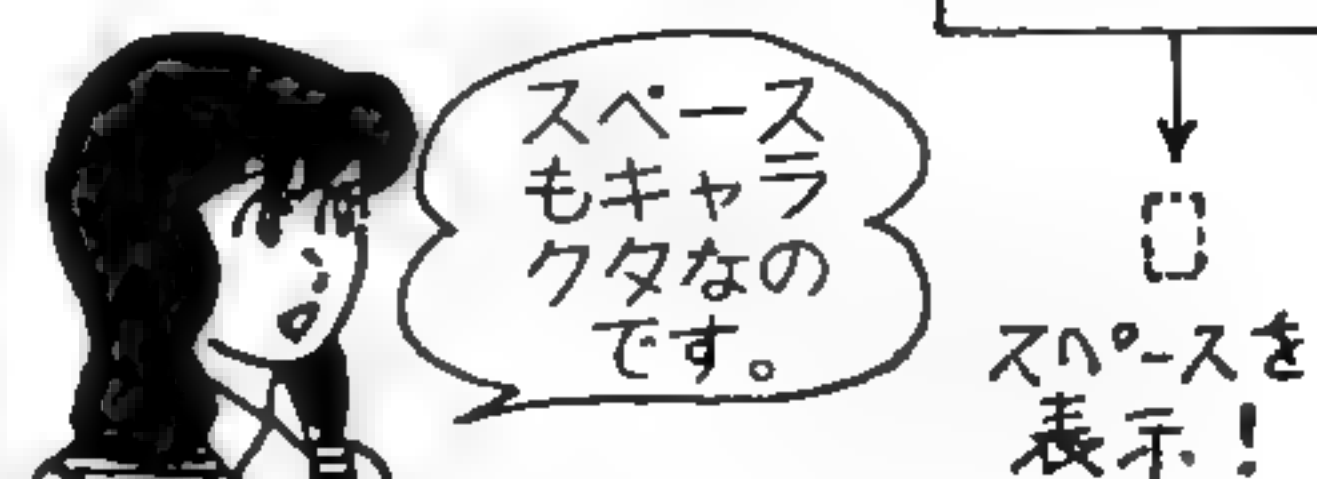


あと2バイト分のデータもこれと同じことをやればよいわけですからこの部分をサブルーチンにします。



区切りのためにフランクを表示します。フランクのためのアスキーコードは20Hです。

```
MVI C, 2
MOV E, 20H
CALL 5
```





ではED(エディタ)を使って  
TOMOKO.ASMにこれらをつ  
つけてみます。



ED TOMOKO.ASM

A>ED TOMOKO.ASM  
:\*

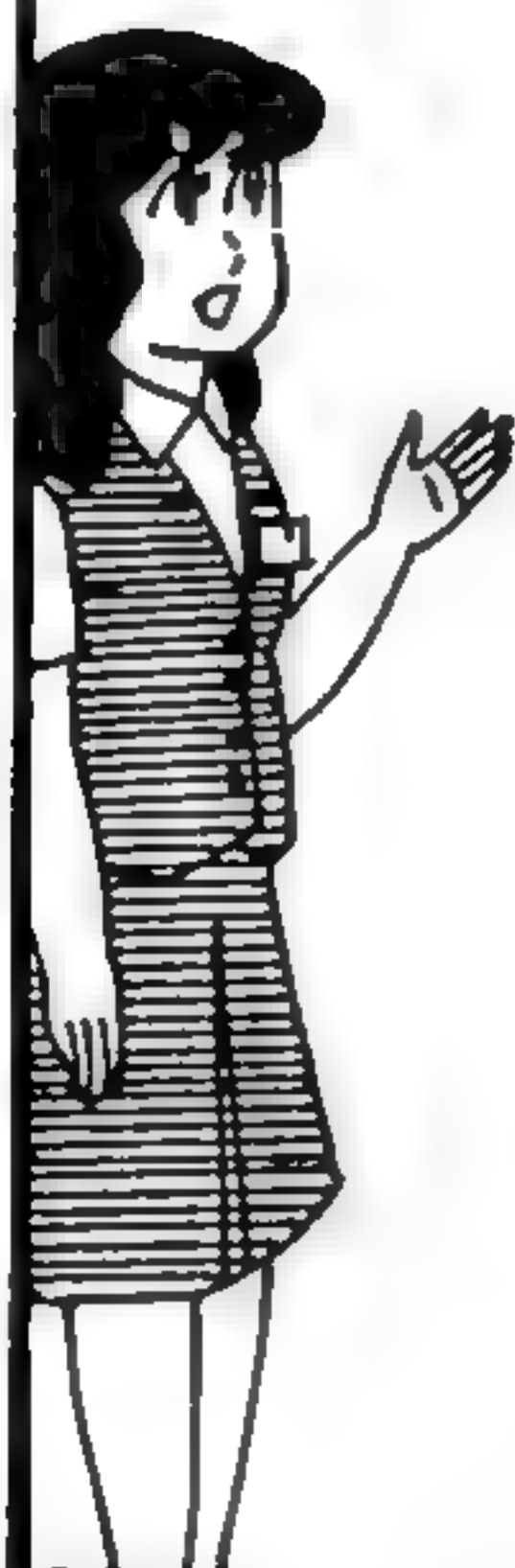
今度はNEW  
FILEとは表示され  
ません。

サブルーチンです。

SATOMI:ANI 0FH

RRC  
RRC  
RRC  
RRC  
ANI 0FH  
ORI 30H  
PUSH B  
MVI C,2  
MOV E,A  
CALL 5  
POP B  
LDAX B  
ANI 0FH  
ORI 30H  
PUSH B  
MVI C,2  
MOV E,A  
CALL 5  
MVI C,2  
MOV E,20H  
CALL 5  
POP B  
RET

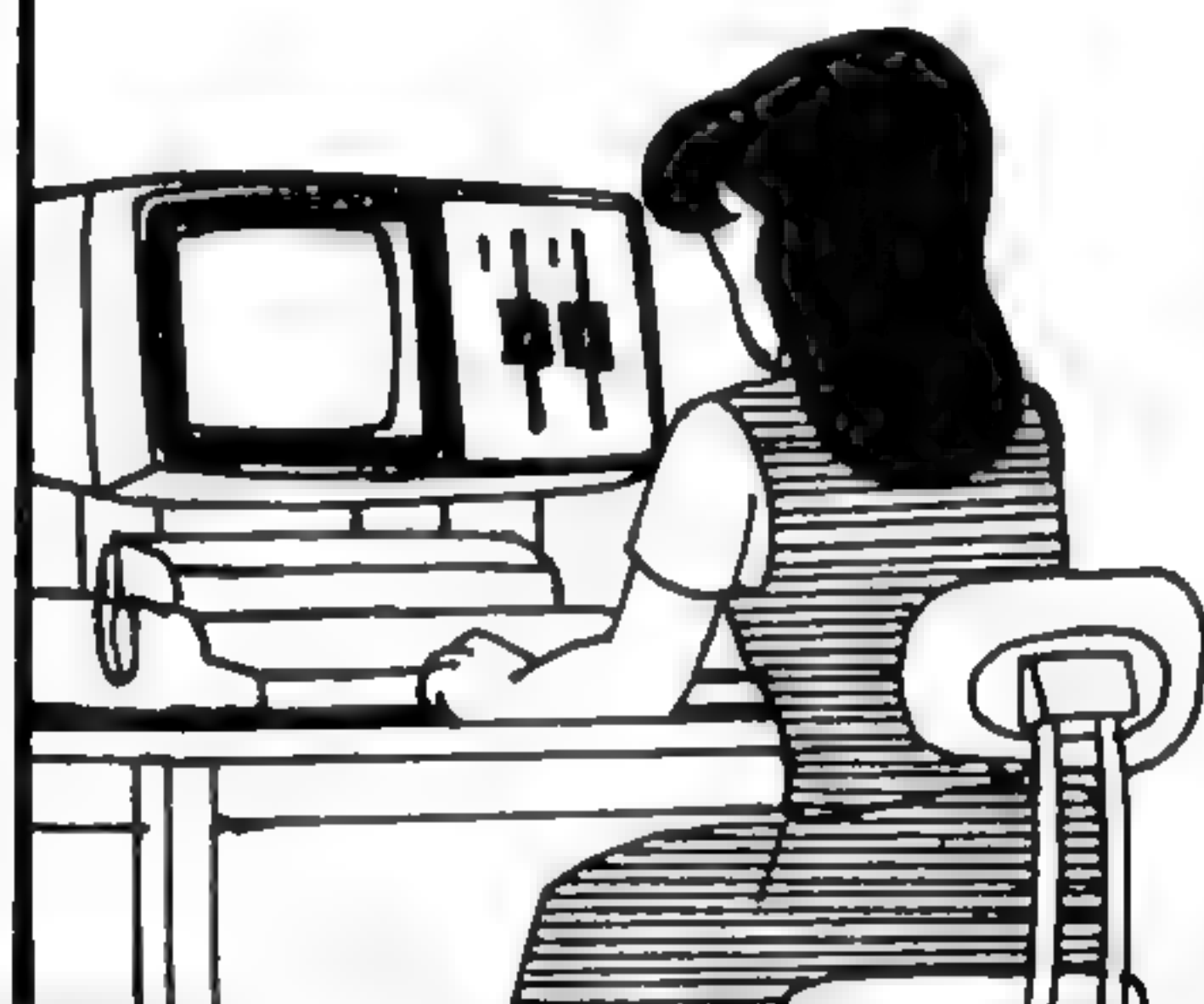
ゼロが  
必要



表示部のメインルーチ  
ンです。

DONE:HLT  
↓  
LXI B,2002H  
LDAX B  
CALL SATOMI  
DCX B  
LDAX B  
CALL SATOMI  
PCX B  
LDAX B  
CALL SATOMI  
RET  
END

CP/Mへ戻  
るためのリターン  
文です。



数字±nを入れて、すると±n  
番目の内容が表示されます。

4: XRA A  
4:\* -3  
1: ORG 100H  
1:\*  
2: LXI B,2000H  
2:\*

-3

1

これでバッファの中にTOMO  
KO.ASMの内容が移ります。  
で順次1行ずつ見ることができ  
ます。

A>ED TOMOKO.ASM  
:\*0A  
1:  
2: LXI B,2000H  
2:\*  
3: LXI H,2003H  
3:\*  
4: XRA A  
4:\*

O(ゼロ)A

このAコマンドは、  
ASMファイルか  
らバッファにデー  
タを移すためのも  
のです。

A>ED TOMOKO.ASM  
:\*0A  
1:

(モニタ表示)

2: LXI B,2000H  
2:\*I  
2: LXI H,2000H  
3: MVI M,57H  
4: INX H  
5: MVI M,28H  
:  
13: MVI M,18H

ORG 100Hの次から入  
れていくことになります。

LXI H,2000H  
MVI M,57H  
INX H  
MVI M,28H  
:  
MVI M,18H

今ORG 100HとLXI  
B,2000Hの間にデータロー  
ドのルーチンを挿入します。イン  
サートモードにするためのコ  
マンドはIです。

2: LXI B,2000H  
2:\*I  
2:

I

この場合、リスト  
でみるとLXI B,  
2000Hの上に挿入  
されていきます。



ひとつ戻してDONE←HLT  
を表示させます。

```
13: MVI M, 18H
14:
   : * - B
28:
28: * - 1
27: DONE ← HLT
27: *
```

±Bコマンドは行の表示をテ  
キストの先頭か末尾に一気  
にとばすものです。

```
13: MVI M, 18H
14:
   : * - B
28:
28: *
```

インサートモードの終了は  
^Zです。[CTRL]キーと[Z]キ  
ーを同時に押すことを意味し  
ます。

```
13: MVI M, 18H
14:
   : *
```

28番からインサートするので、  
ひとつ空振りします。↵

```
27: DONE : LXI B, 2002H
27: *
28:
28: * I
28:
```

28番には何も入っ  
ていません。

確かめます。-1↵

```
27: * SHLT ^Z 2002H
28:
28: * - 1
27: DONE : LXI B, 2002H
27: *
```

DONE←HLTのHLTを  
LXI←B, 2002Hに置き換え  
ます。Sコマンドをこのよう  
に 使います。

```
27: DONE ← HLT
27: * SHLT ^Z LXI ← B, 2002H
28:
28: *
```

SHLT ^Z KXI ← B,  
2002H↵

カチャ カチャ

^Zでインサートモード終了  
します。

```
61: RET
62:
   : *
```

^Z

(モニタ表示)

```
28: * I
28: LDAX B
29: CALL SATOMI
   :
37: END
38: SATOMI: ANI OFOH
39: RRC
   :
61: RET
62:
```

以下前ページのプログラムの  
順に入れていきます。

```
LDAX B ↓
CALL SATOMI ↓
   :
END ↓
SATOMI: ANI OFOH ↓
RRC ↓
   :
RET ↓
```

FDD上のTOMOKO.ASM  
には今編集したテキストが入り、  
TOMOKO.BAKには編集前  
のテキストがバックアップ用と  
して入るのです。

```
61: RET
62:
   : * E
```

A>↵

Eコマンドで今編集したテキス  
トをFDに書き込みます。

```
61: RET
62:
   : * E
```

E↵

カチャ

カチャ

今は使いませんでしたが行を  
削除したい場合もありますね。  
この時はKコマンド(キル)を  
使います。

K↵など



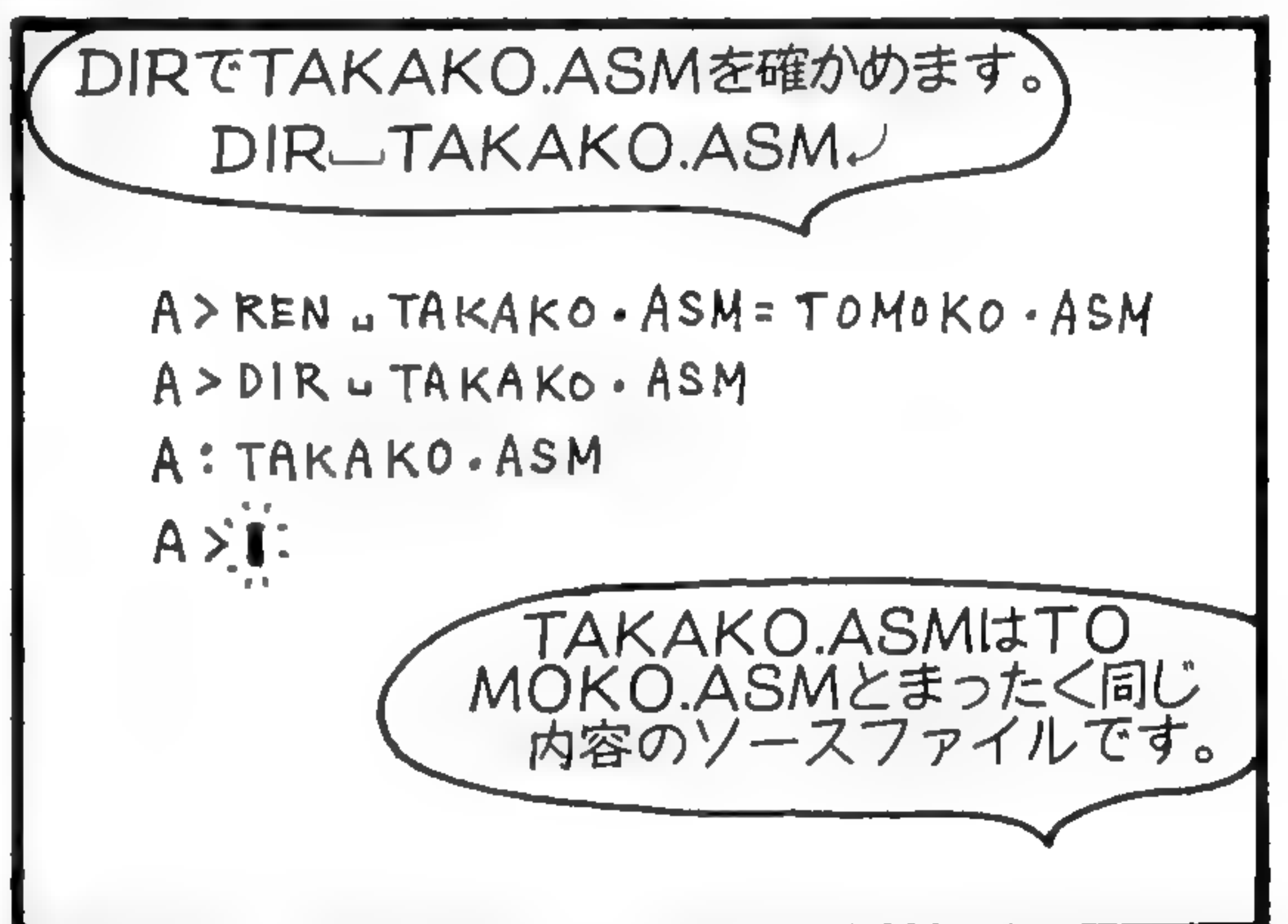
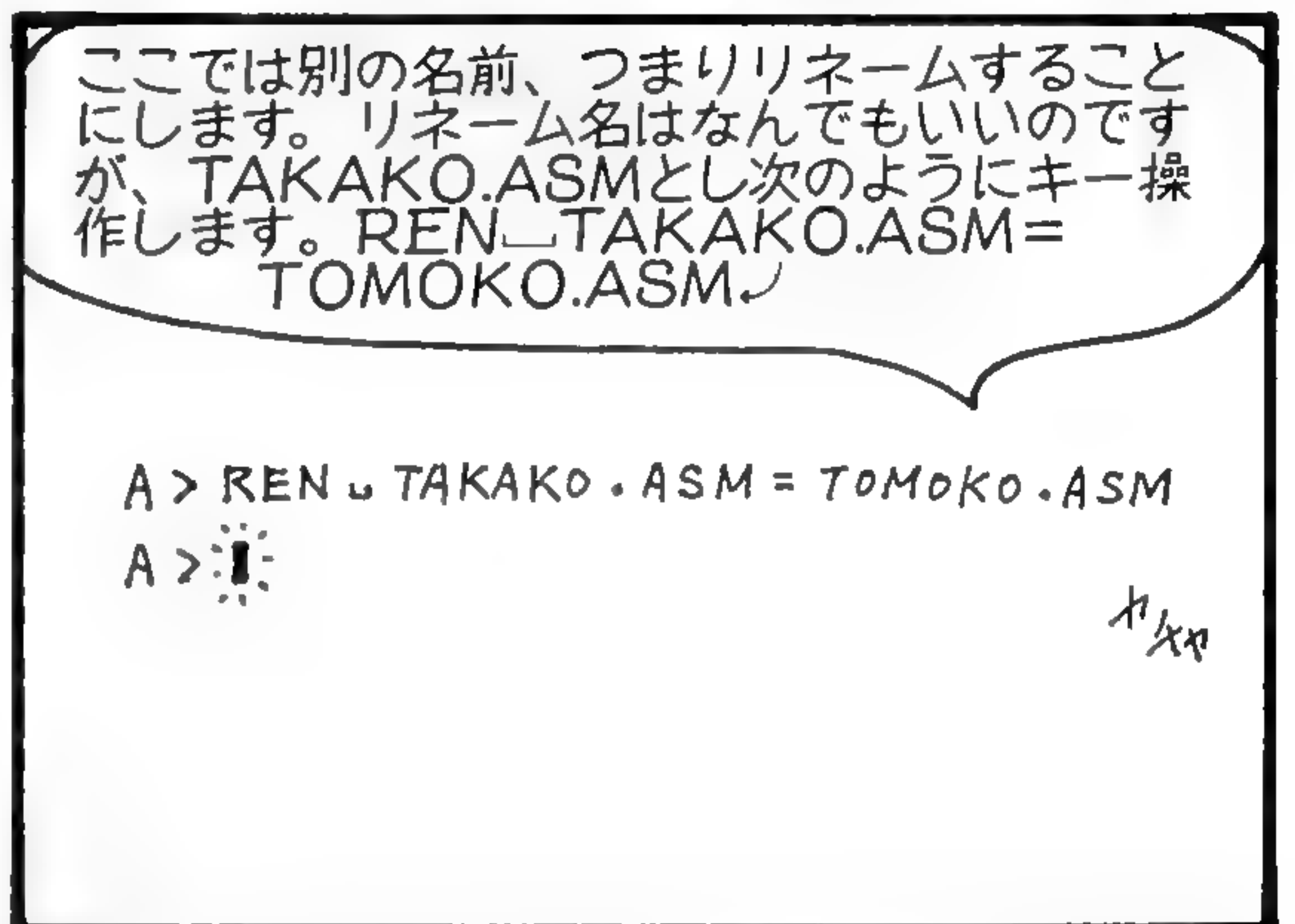




```

0100          ORG 100H
0100 210020    LXI H,2000H
0103 3657      MVI M,57H
0105 23        INX H
0106 3628      MVI M,28H
0108 23        INX H
0109 3634      MVI M,34H
010B 23        INX H
010C 3618      MVI M,18H
010E 23        INX H
010F 3639      MVI M,39H
0111 23        INX H
0112 3618      MVI M,18H
0114 010020    LXI B,2000H
0117 210320    LXI H,2003H
011A AF        XRA A
011B 1E03      MVI E,03H
011D 0A        LOOP:LDAX B
011E 8E        ADC M
011F 27        DAA
0120 02        STAX B
0121 1D        DCR E
0122 CA2A01    JZ DONE
0125 03        INX B
0126 23        INX H
0127 C31D01    JMP LOOP
012A 010220    DONE:LXI B,2002H
012D 0A        LDAX B
012E CD3C01    CALL SATOMI
0131 0B        DCX B
0132 0A        LDAX B
0133 CD3C01    CALL SATOMI
0136 0B        DCX B
0137 0A        LDAX B
0138 CD3C01    CALL SATOMI
013B C9        RET
013C E6F0      SATOMI:ANI 0F0H
013E 0F        RRC
013F 0F        RRC
0140 0F        RRC
0141 0F        RRC
0142 E60F      ANI 0FH
0144 F630      ORI 30H
0146 C5        PUSH B
0147 0E02      MVI C,2
0149 5F        MOV E,A
014A CD0500    CALL 5
014D C1        POP B
014E 0A        LDAX B
014F E60F      ANI 0FH
0151 F630      ORI 30H
0153 C5        PUSH B
0154 0E02      MVI C,2
0156 5F        MOV E,A
0157 CD0500    CALL 5
015A 0E02      MVI C,2
015C 1E20      MVI E,20H
015E CD0500    CALL 5
0161 C1        POP B
0162 C9        RET
                END

```





ではTAKAKO、

たし算実行結果の答えです。

```
A>TAKAKO
52 67 75
A>
```

TAKAKO、とすればCP/MがTAKAKO.COMファイルをFDDから読み出しメモリにロードします。ロード完了後プログラムが走ります。

ファイルを確認します。  
DIR、TAKAKO.\*、

```
A>DIR、TAKAKO.*
A: TAKAKO ASM: TAKAKO BAK
: TAKAKO PRN: TAKAKO HEX
: TAKAKO COM
A>
```

でも、このカンタンなプログラムでもちゃんと表示するまでいくつかのデバッグが必要でした。

誰が何と言おうと私のプログラムもCP/Mのうえを走ったのです。

あはっ、やった。マシン語のプログラムが走った。

52、67、75

このDDTにもいくつかのコマンドがあります。まずアドレス2000から200Fまで0(ゼロ)にします。F(フル)コマンドを使います。F2000, 200F, 0、

```
NEXT PC
0163 0000
-F2000, 200F, 0
```

答の入るところをクリアするため。

DDTが扱うファイル形式はHEXです。DDT、TAKAKO.HEX、とキーをたたきます。

```
A>DDT、TAKAKO.HEX
DDT、VERS 2.2
NEXT PC
0163 0000
```

DDTのフロンフトです。

ニーモニックプログラムを二ラミながらのデバック作業は大変です。このためCP/MではDDTというデバッガがあります。

この結果2000~2005にたし算の問題の数値57、28、34、18、39、18が入っていればこの部分のプログラムはOKということになります。  
D2000, 200F、

```
-G100, 114
*0114
-D2000, 200F
2000 57 28 34 18 39 18 00
```

OKです。

TAKAKO.PRN(前ページ)を見てください。アドレス100から114までプログラムを走らせます。G(ゴー)コマンドを使います。G100, 114、

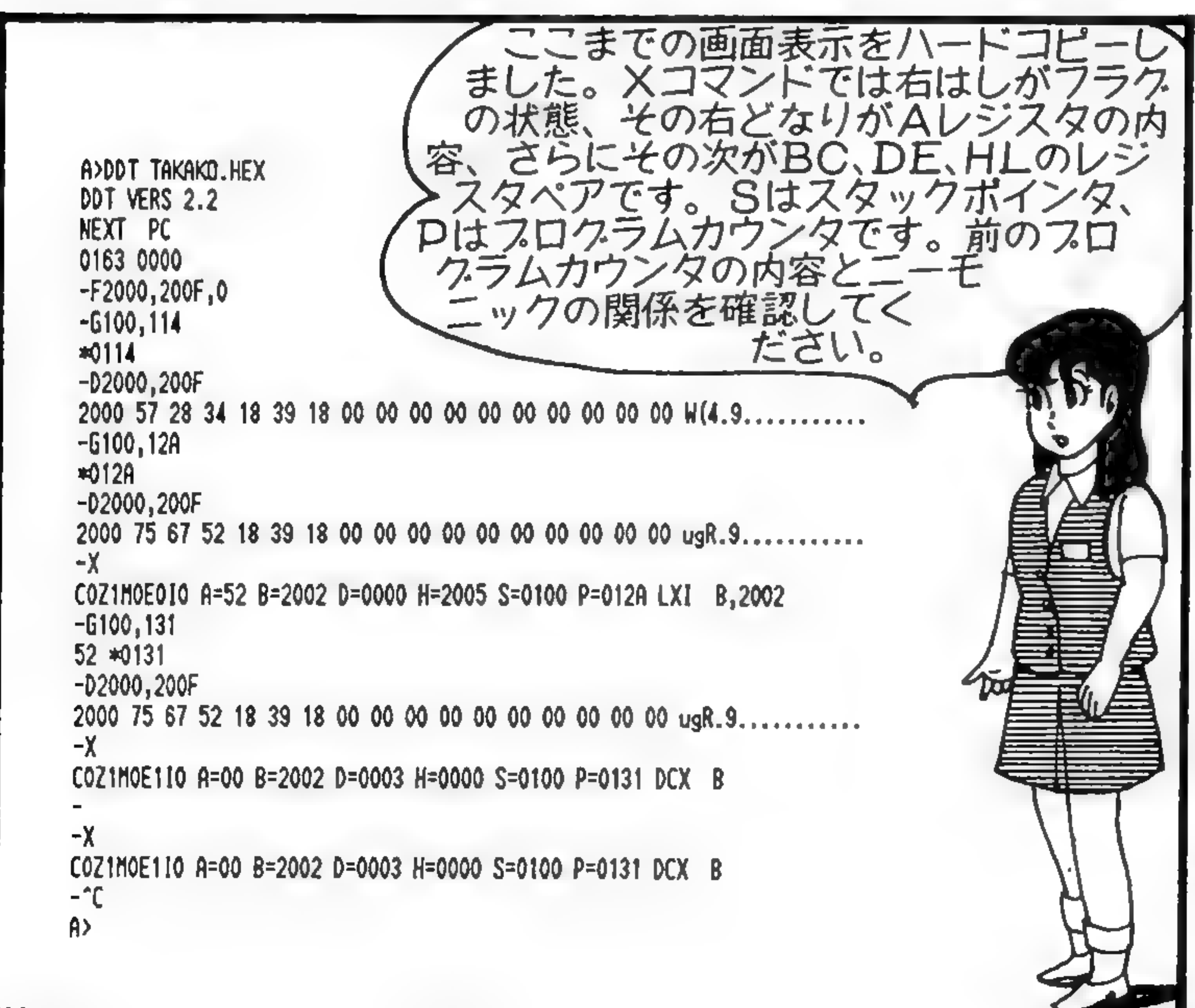
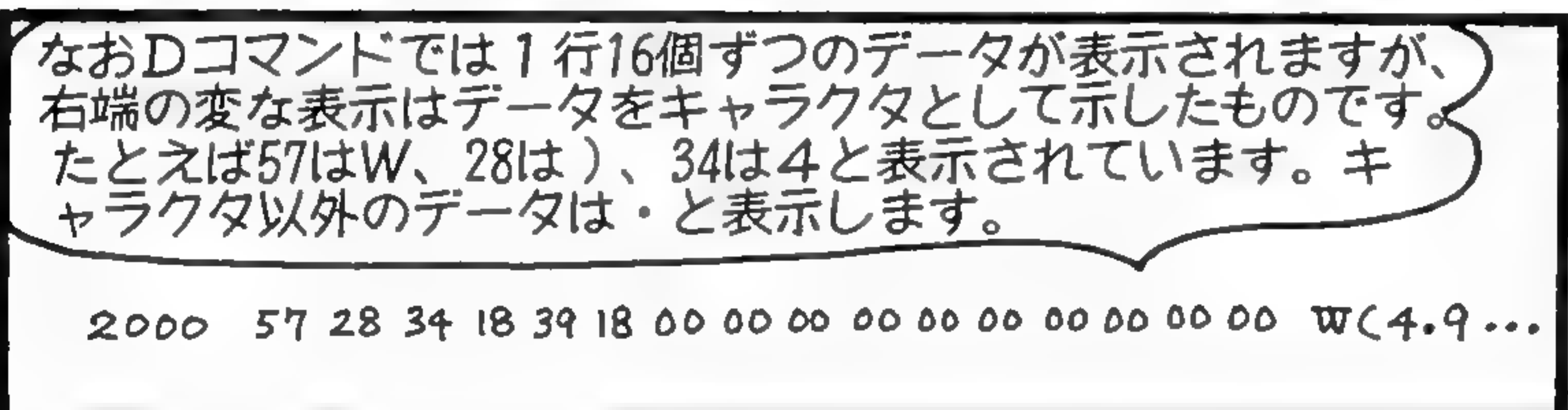
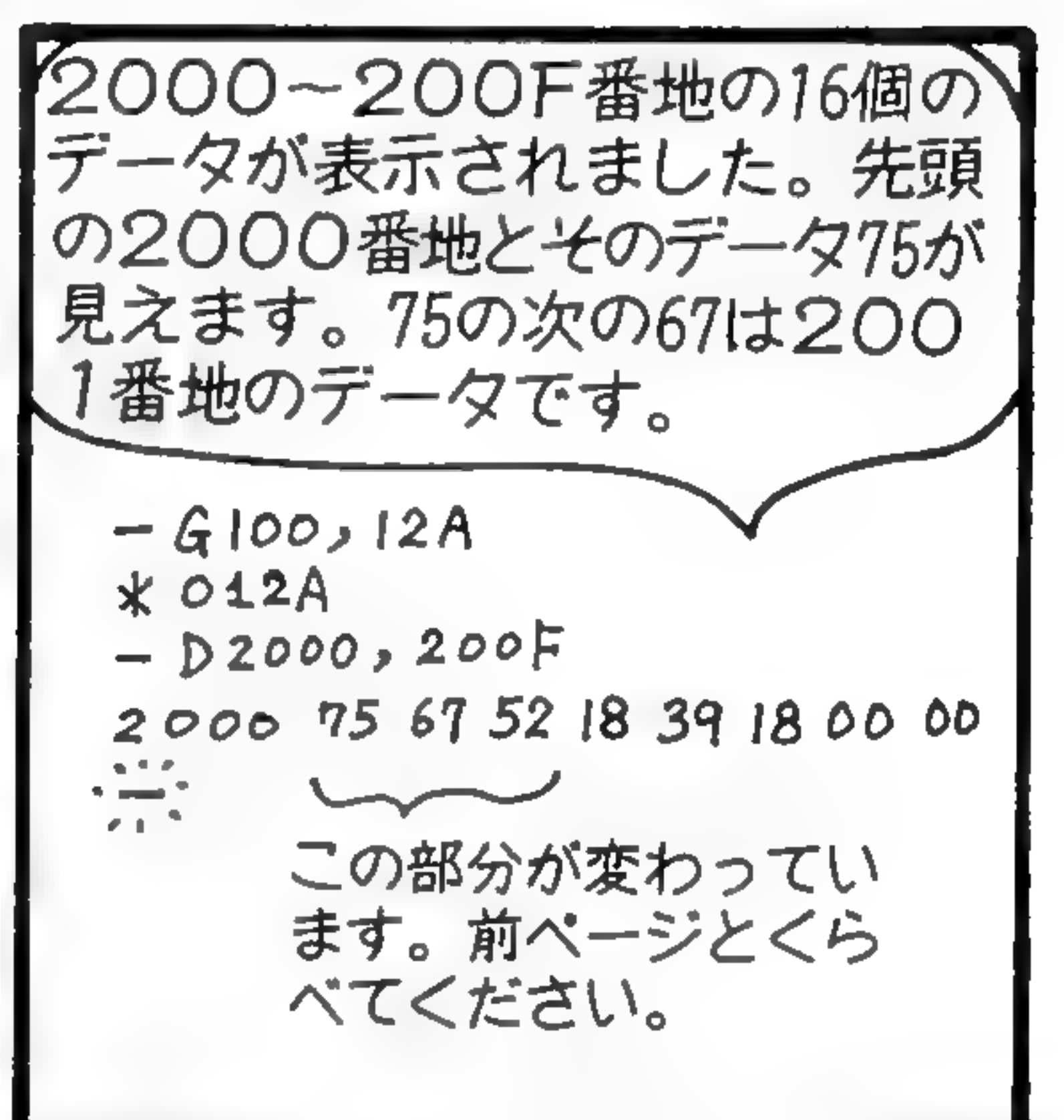
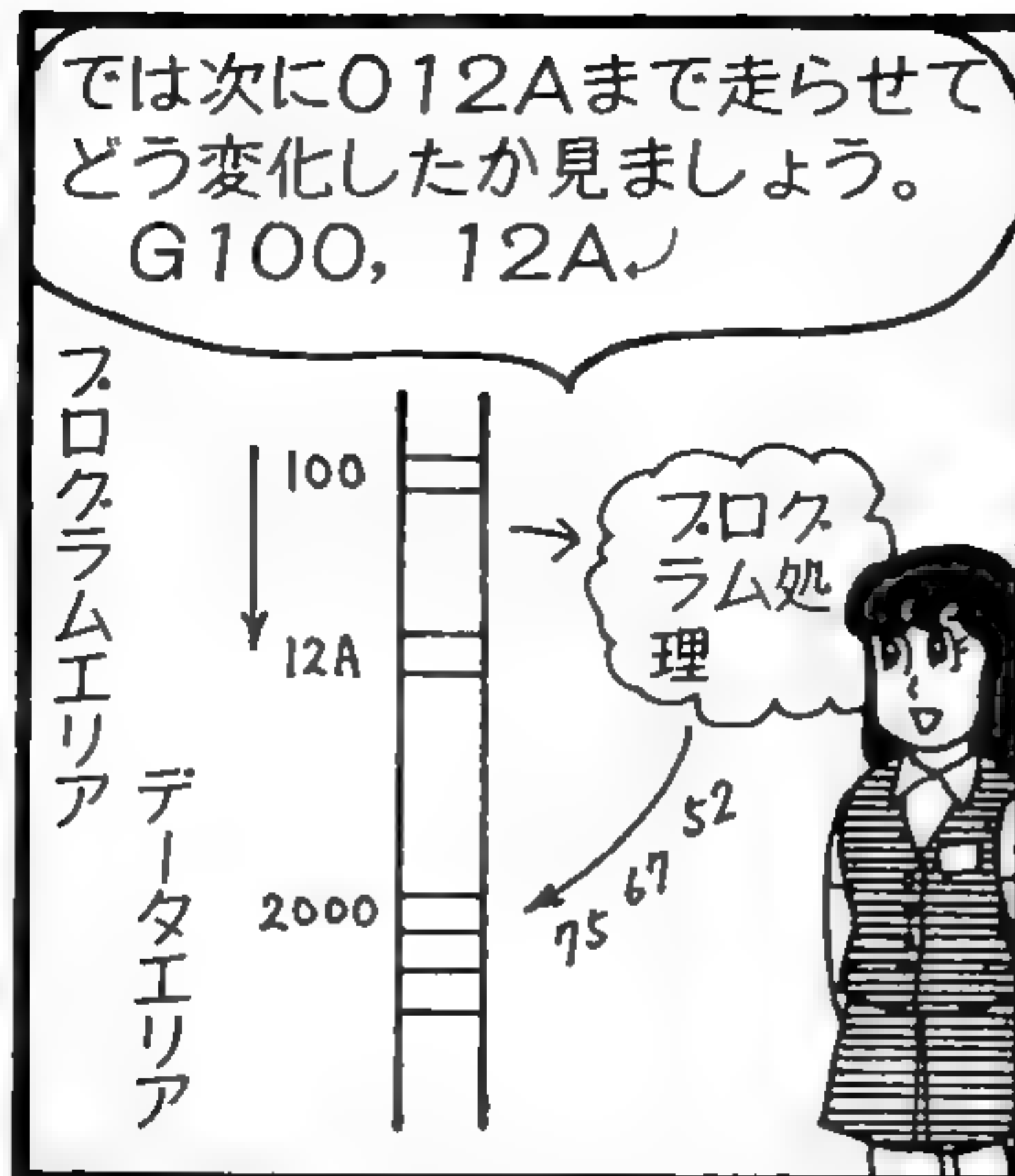
```
2000 .00 00 00 00 00 00 00
-G100, 114
*0114
```

メモリの内容を見たい時にはD(ダンプ)コマンドがあります。D2000, 200F、

```
-F2000, 200F, 0
-D2000, 200F
2000 00 00 00 00 00 00 00
```

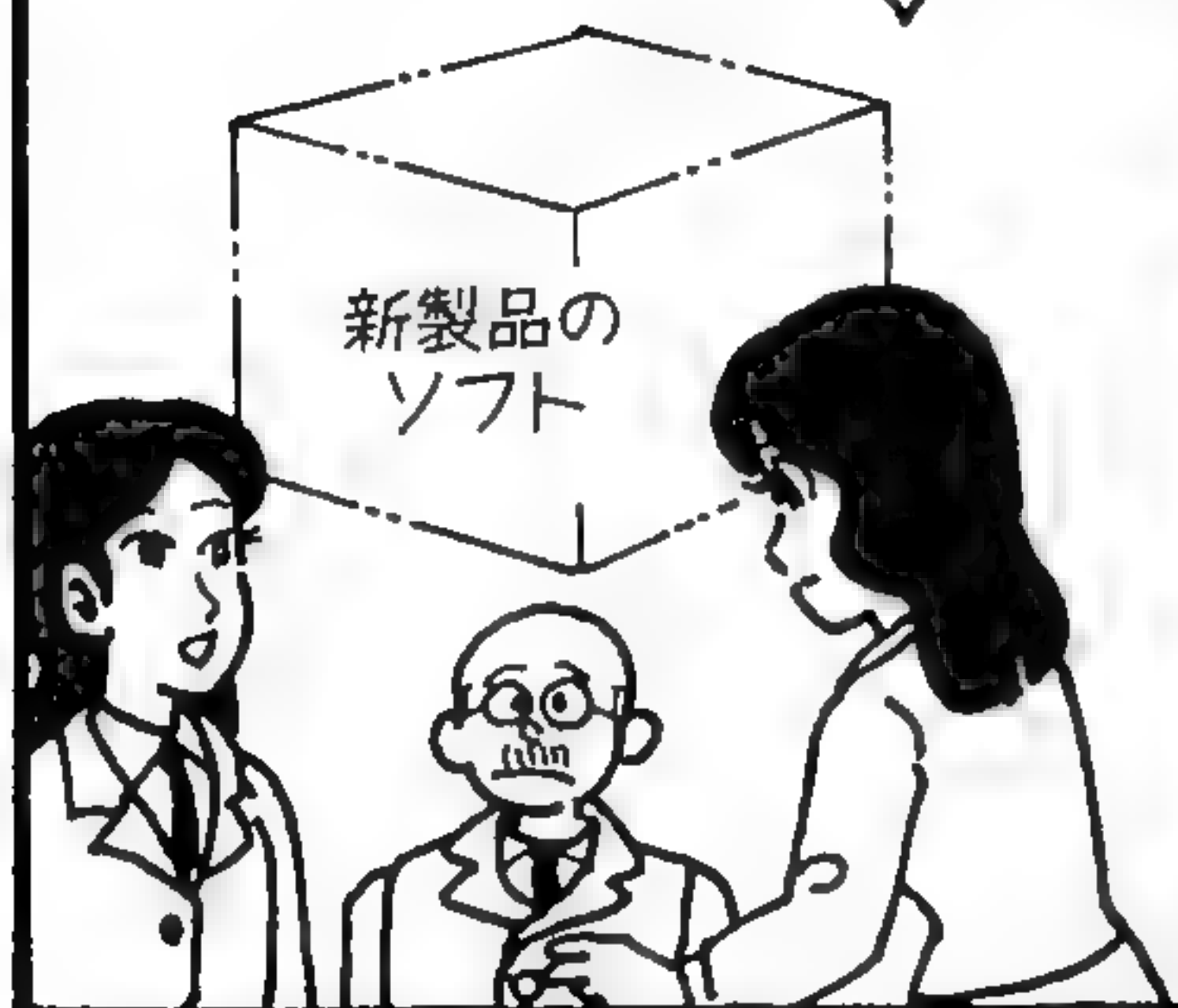
2000番地のメモリの先頭アドレス 内容







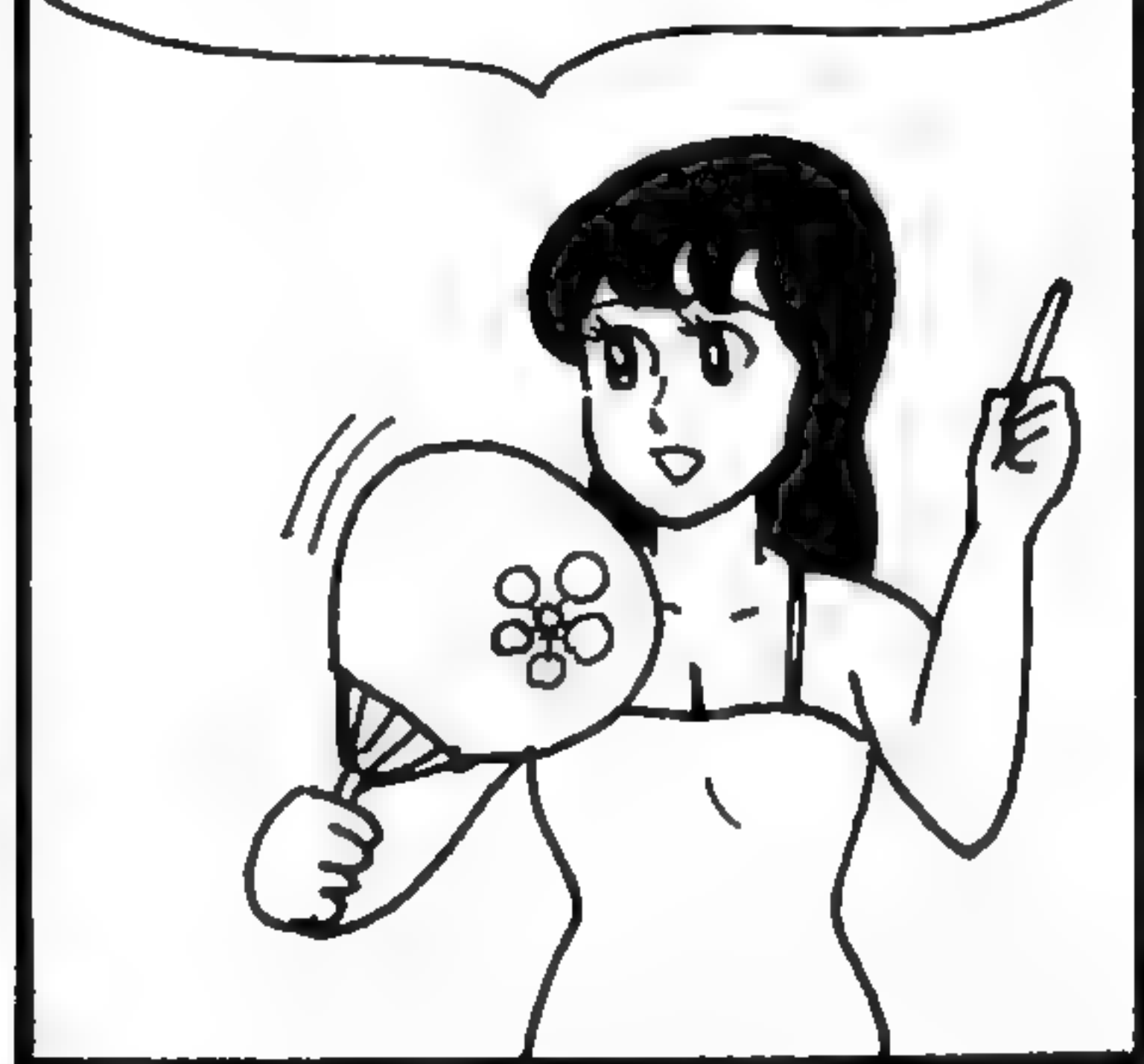
CP/Mは新しいハード構成のパソコンを作る時、またそのパソコン用のBASICインタプリタを作る時などに威力を発揮することになるでしょう。



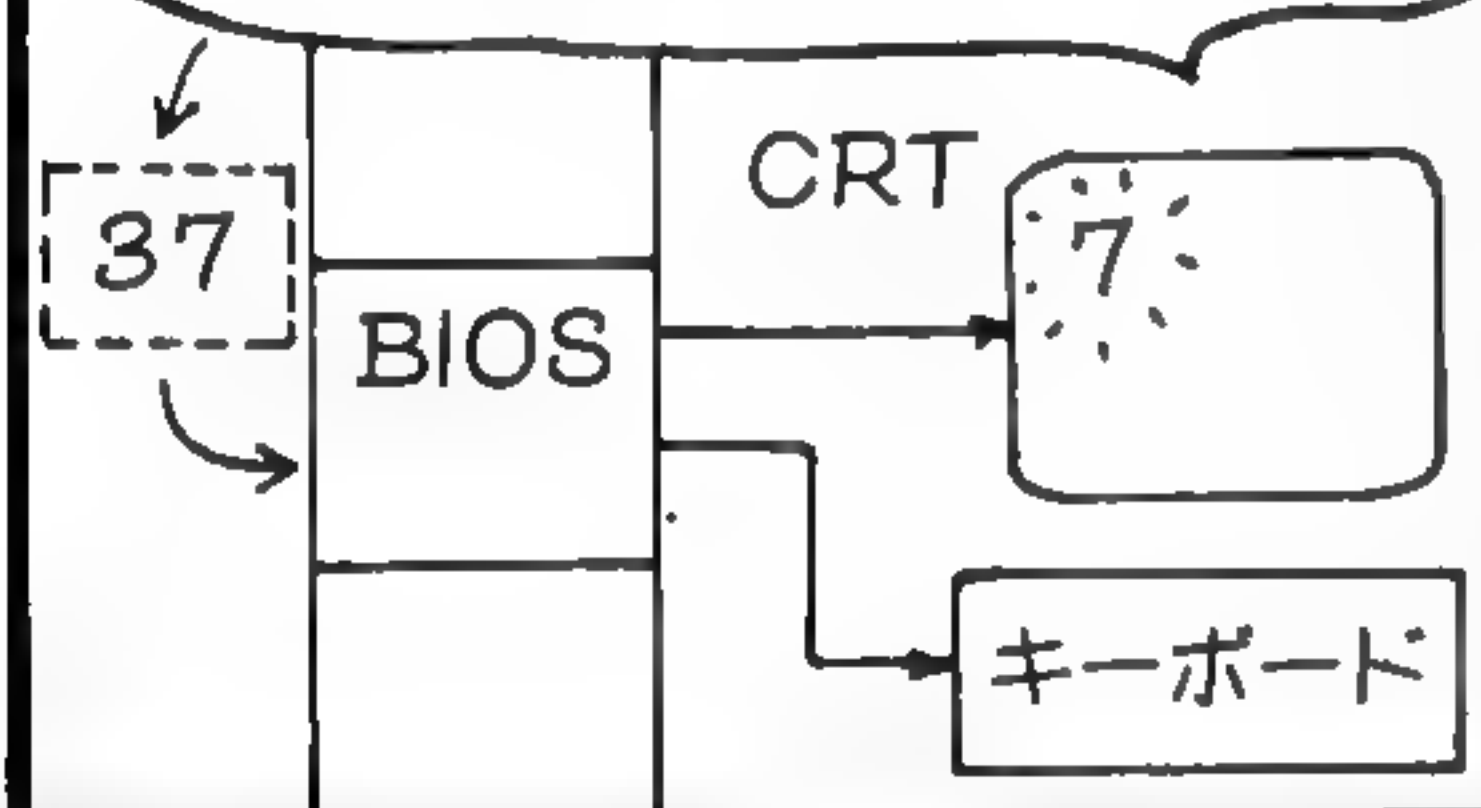
実は私たちがBASICや簡易言語でパソコンを操作している時は、CP/Mは全然表に出てこないのです。



CP/M、CP/Mと騒いだわりにはそれらしいものがあまり出てきませんでした。いったいどこがCP/Mなのでしょう。



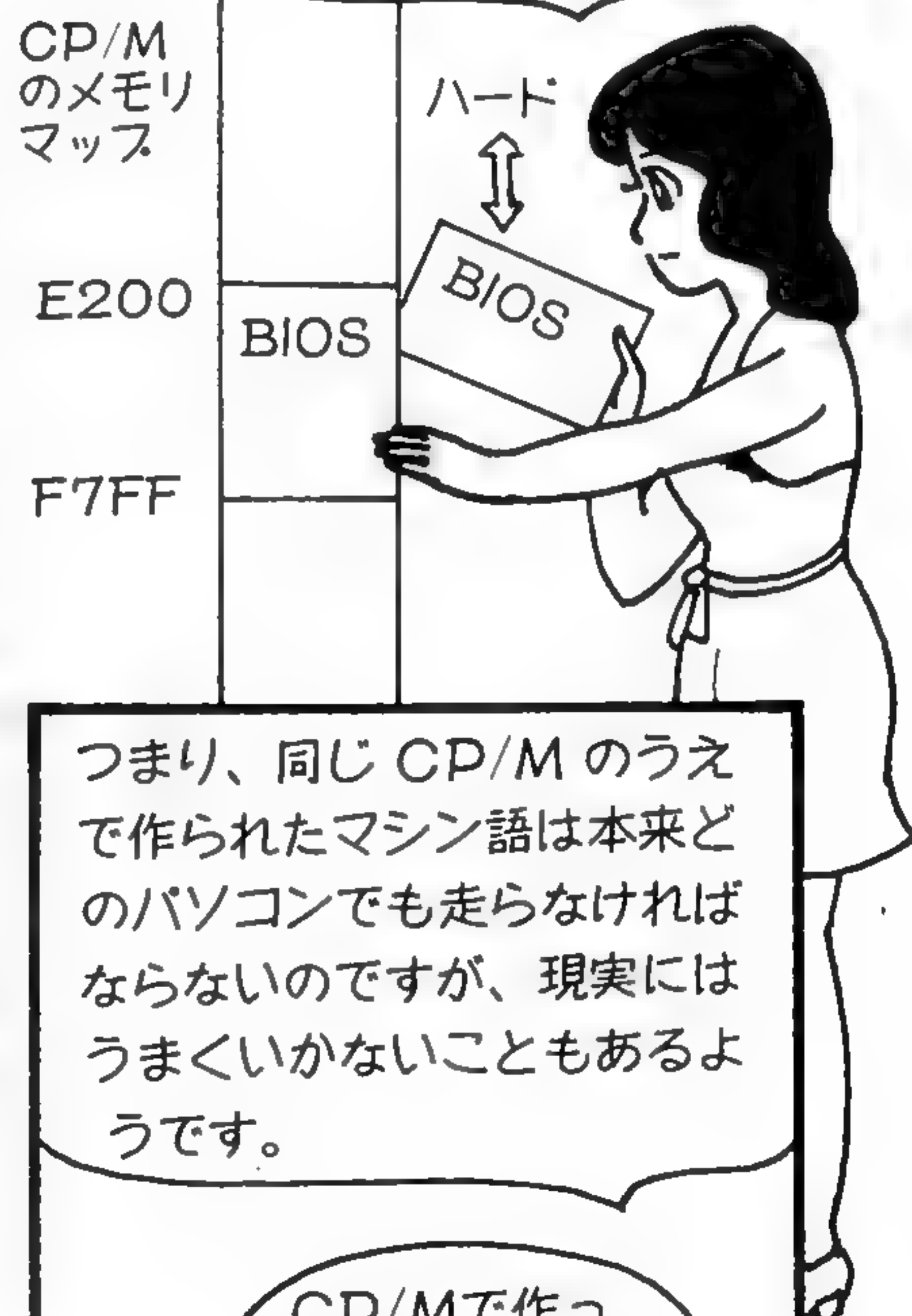
このシステムコールのファンクションの中にはI/Oのハードに関するプログラムはないのです。つまり、ハードの構成は本文では受け持たずアドレス5から始まるサブルーチンがハード選択の機能を持っていることがわかります。



今のプログラムで1キャラクタを表示するのに、システムコールの2番を使いましたね。



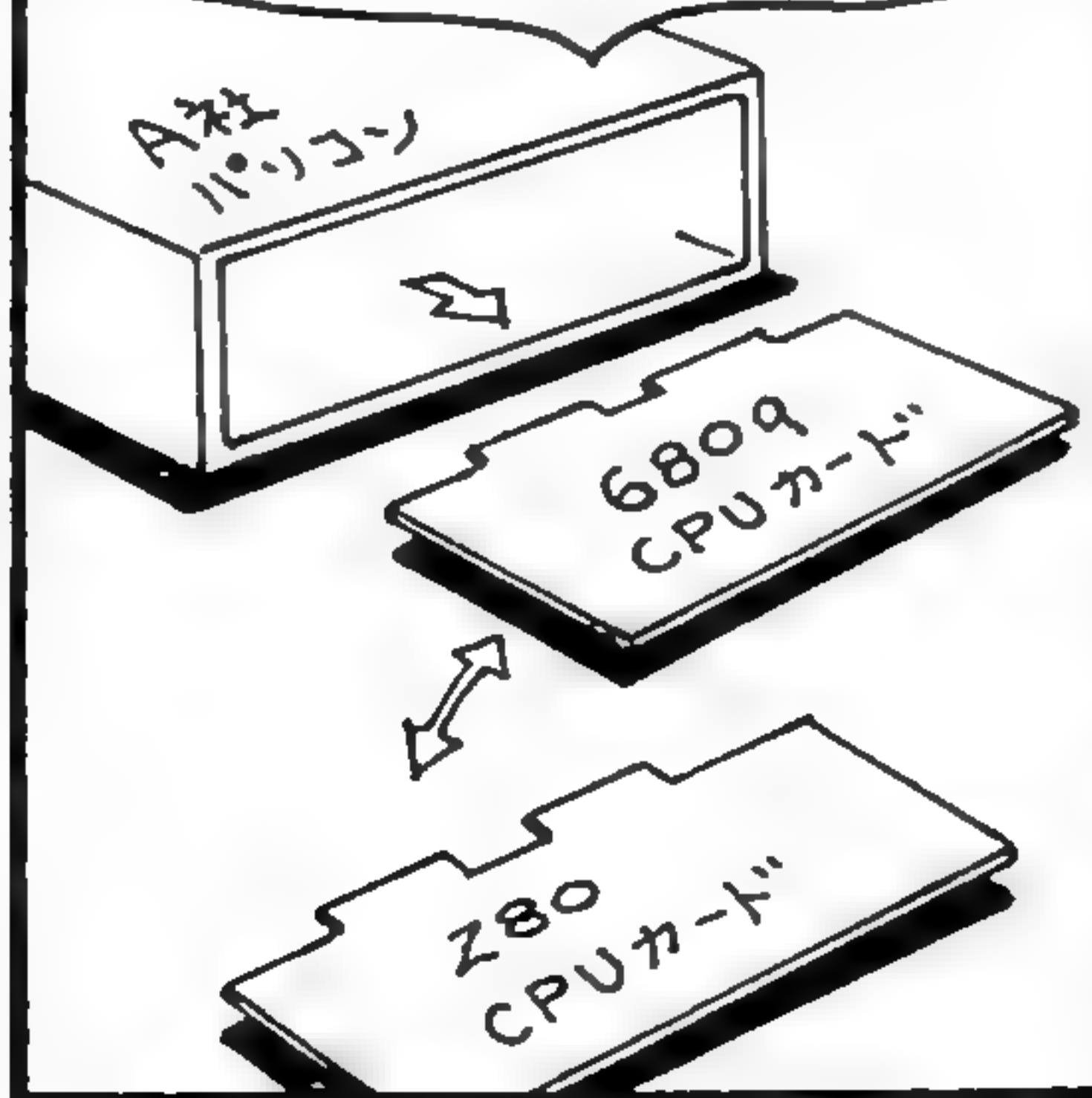
それまでに作ったインタプリタ自身は変えることなくCP/MのI/Oに関係のあるブロックだけを変更すればよいのです。



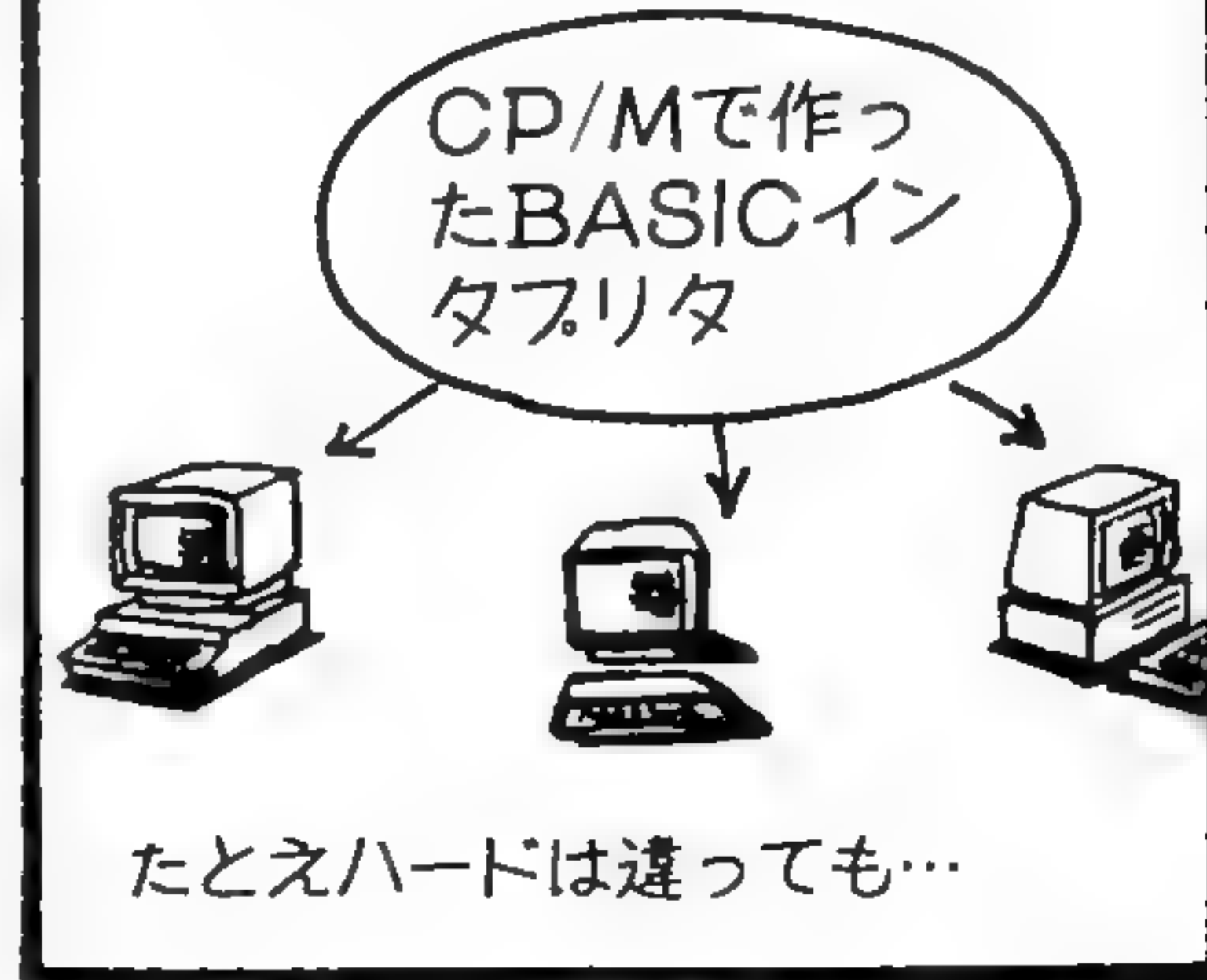
またこの時注意が必要なのはA社のパソコンのA社のZ80カードと変換し、A社のそのパソコン用のCP/Mでないとダメということです。B社のZ80カードとC社のCP/Mでやろうとしても無理です。



また、80系以外のパソコンでCP/Mを走らせるためには、CPUカードをZ80のついたカードに変換する必要があります。



つまり、同じCP/Mのうでで作られたマシン語は本来どのパソコンでも走らなければならないのですが、現実にはうまくいかないこともあるようです。






③ Z80アセンブラを試す

Z80のアセンブラには少なくともM80とL80が必要です。システムディスクにコピーします。

A>DIR  
.....M80.COM.....  
.....L80.COM.....  
.....ZSID.COM.....

M80、L80などはオプションです。

.MACファイルをつくります。



L80.COMもZSID.COMも同様にコピーします。

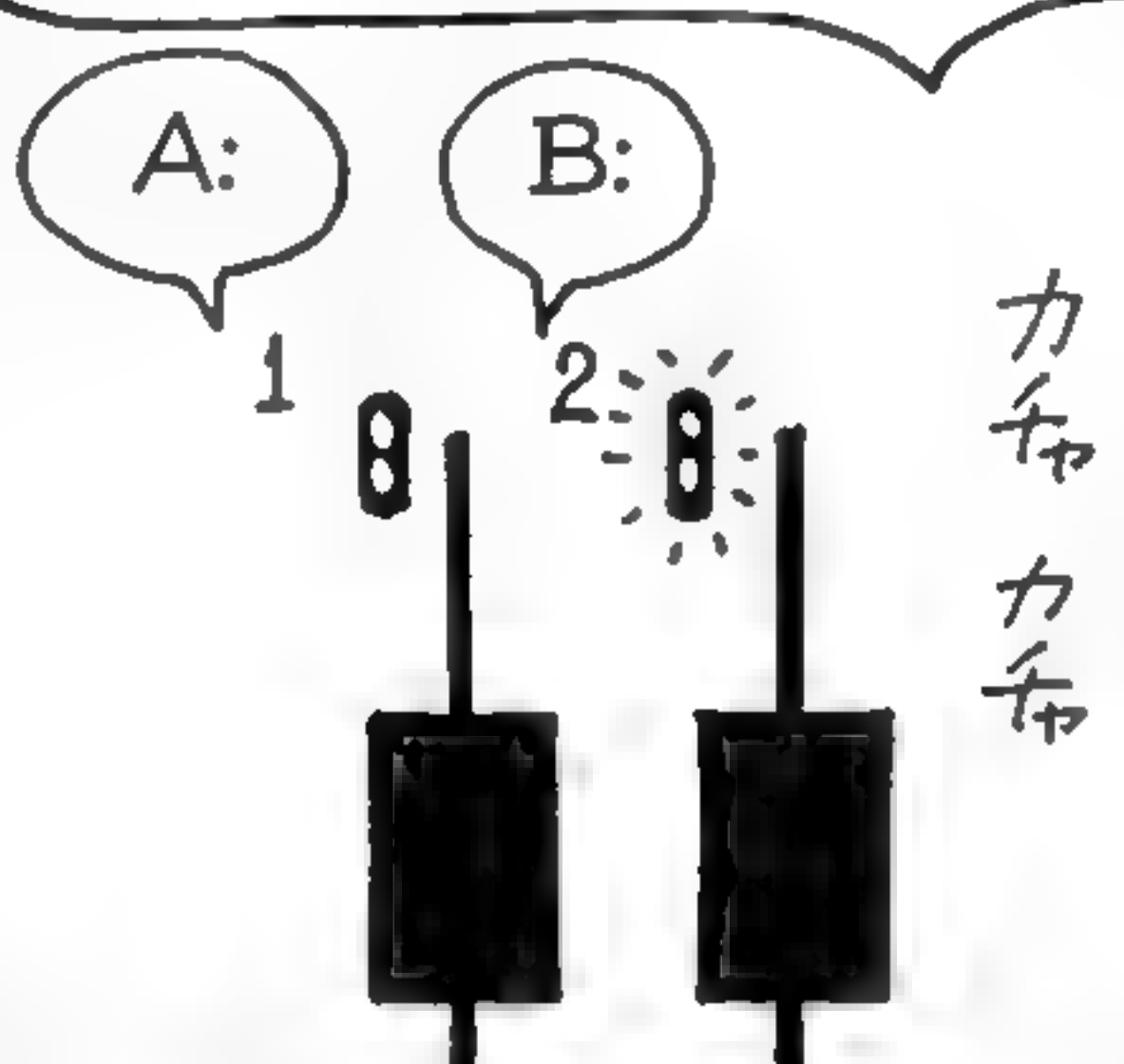
PIP A:=B:80.COM✓  
PIP A:=B:ZSID.COM✓

A>PIP A:=B:M80.COM  
A>PIP A:=B:L80.COM  
A>PIP A:=B:ZSID.COM

これでB:に入っているM80.COMファイルがA:のFDにコピーされます。

A: 1  
B: 2

カチ  
カチ



コピーはシステムFDをB:に入れ自分のをA:に入れます。自分のにもCP/Mのシステムが入っていないかもしれません。

PIP A:=B:M80.COM✓

キーイン  
↓ 表示

A>PIP A:=B:M80.COM

ソースファイルはエディタで作ります。まず、A>の状態にします。



前と同じプログラムをZ80のニーモニックに直してソースファイルを作り、それをアセンブルしてみます。



エディタ以外は使用するプログラムや拡張子が異なります。

ニーモニック ファイル	8080	Z80
エディタ	ED.COM	ED.COM
アセンブラ	ASM.COM	M80.COM
ローダ	LOAD.COM	L80.COM
デバッガ	DDT.COM	ZSID.COM
拡張子	.ASM	.MAC

ここで少しめんどろなのですが、次のコマに書いているような内容をつけ加えたプログラムにする必要があります。

ASEG✓  
ORG 0100H

:\*|  
1: ASEG  
2: ORG 0100H  
3:

次に、修正などを行なうなら #A✓としてファイルをメモリに移しますが、この場合おニューなのでこのまま1コマンドでプログラムを書き込んでいきましょう。1✓

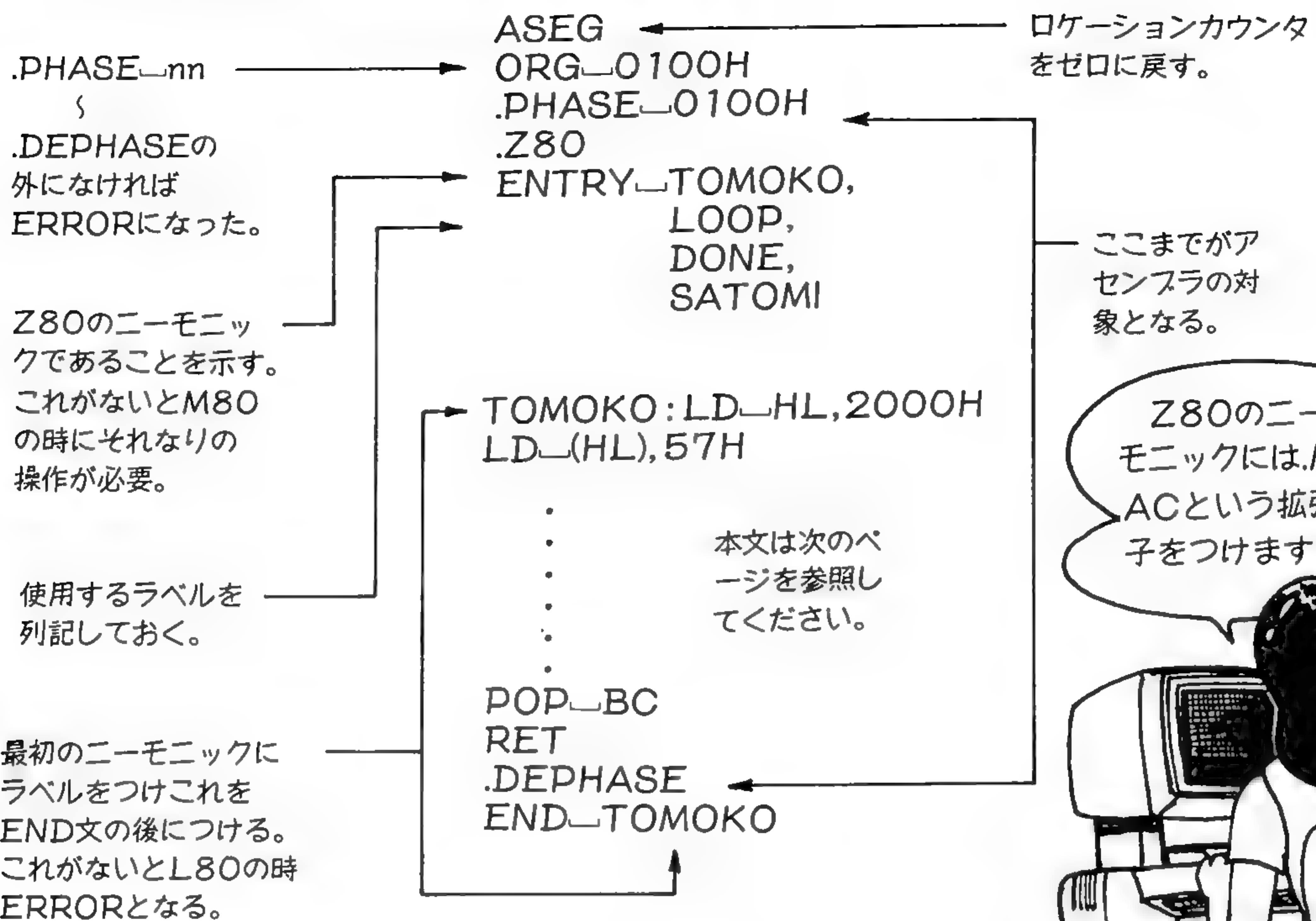
NEW FILE  
:\*|  
1:

次にED HITOMI.MAC✓とキーインします。HITOMIがこのファイルの名前です。必ず.MACをつけておかなければあとでアセンブルできません。

A>HITOMI.MAC  
NEW FILE  
:\*



## .MACファイル作成時の注意事項



アセンブルファイルの作成とリストアップのプリントをするコマンドです。  
HITOMI, LST:=HITOMI✓

A>M80  
\*HITOMI, LST:=HITOMI



HITOMI.MACはTAKAKO.ASMのニーモニックを変えただけですがそれでもエラーはありました。これは修正完了後のHITOMI.MACです。アセンブルはM80✓

A>M80

\*



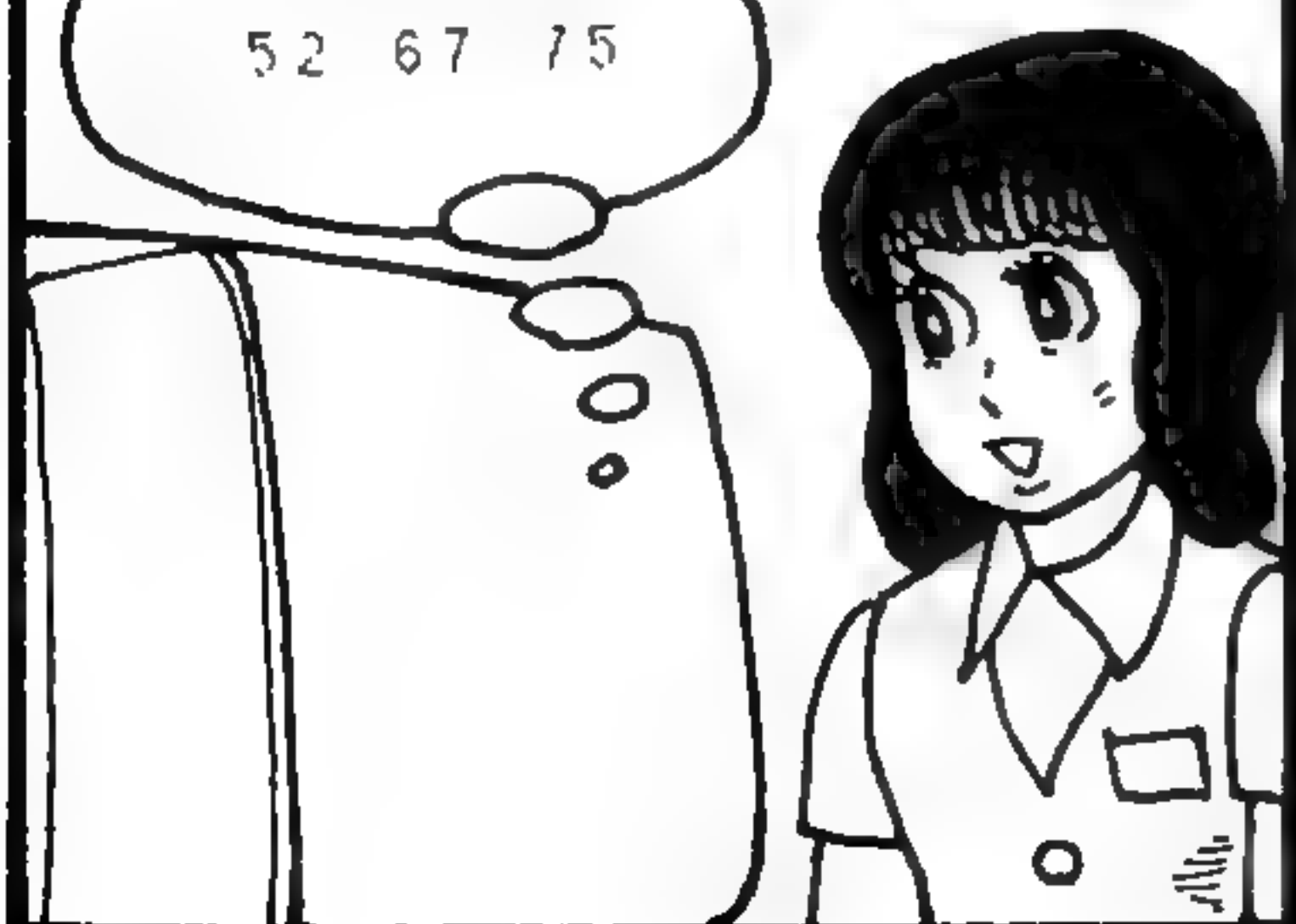
エディタの終わりはコマンドE。これでFDにHITOMI.MACファイルが作成されます。E✓

71: \*  
71: END TOMOKO  
72: \*E



ではこのプログラムを走らしましょう。HITOMI✓OKです。

A>HITOMI  
52 67 75



ディレクトリコマンドでHITOMIファイルを見てみましょう。

A>DIR HITOMI.\*  
A: HITOMI BAK  
  
: HITOMI MAC : HITOMI  
  
: HITOMI REL  
G: HITOMI COM

続いてリンカーにかけます。L80✓続いてのコマンドはHITOMI/E, HITOMI/N✓です。これで.COMファイルができます。

A>L80  
Link-80 Release 3.3  
\*HITOMI/E, HITOMI/N





0000

```

;F ILC NAME
;HITOMI.MAC
ASEG;Z80 ASM.
ORG C100H
.PHASE C100H;Z80 ASM
.Z80
ENTRY TOMOKO,LOOP,DONE,SATOMI
TOMOKO:LD HL,2000H
0100 21 2000 LD (HL),57H
0101 36 57 INC HL
0102 23 LD (HL),20H
0103 23 INC HL
0104 36 34 LD (HL),34H
0105 23 INC HL
0106 36 10 LD (HL),10H
0107 23 INC HL
0108 36 30 LD (HL),30H
0109 23 INC HL
0110 36 10 LD (HL),10H
0111 36 2000 LD BC,2000H
0112 21 2003 LD HL,2003H
0113 AF XOR A
0114 1E 03 LD E,03H
0115 0A LOOP:LD A,(BC)
0116 8E ADC A,(HL)
0117 27 DAA
0118 02 LD (BC),A
0119 1D DEC E
0120 CA 012A JP Z,DONE
0121 03 INC BC
0122 23 INC HL
0123 C3 011D JP LOOP
0124 01 2002 DONE:LD BC,2002H
0125 0A LD A,(BC)
0126 CD 013C CALL SATOMI
0127 0B DEC BC
0128 0A LD A,(BC)
0129 CD 013C CALL SATOMI
0130 0B DEC BC
0131 0A LD A,(BC)
0132 CD 013C CALL SATOMI
0133 C9 RET;RETURN TO CP/M
0134 E6 FC SATOMI:AND 0FH
0135 0F RRCA
0136 0F RRCA
0137 0F RRCA
0138 E6 0F AND 0FH
0139 F6 30 OR 30H
0140 C5 PUSH BC
0141 0E 02 LD C,2
0142 5F LD E,A
0143 CD 0005 CALL 5
0144 C1 POP BC
0145 0A LD A,(BC)
0146 E6 0F AND 0FH
0147 F6 30 OR 30H
0148 C5 PUSH BC
0149 0E 02 LD C,2
0150 5F LD E,A
0151 CD 0005 CALL 5
0152 0E 02 LD C,2
0153 1E 20 LD E,20H
0154 CD 0005 CALL 5
0155 C1 POP BC
0156 09 RET;RETURN TO HITOMI
0157 .DEPHASE;Z80 ASM
0158 END TOMOKO

```

```

-----JP LOOP
DONE:LD BC,2002H ←
-----LD A,(BC)

```

このようなソースプログラムの方が見やすいことは確かです。アセンブルには関係ありませんが。

これがZ80のニーモニックで作った.RELファイルです。おや? マシン語のアドレスが変ですね。これはリストが見やすいようにしているだけです。

LD BC,2000H

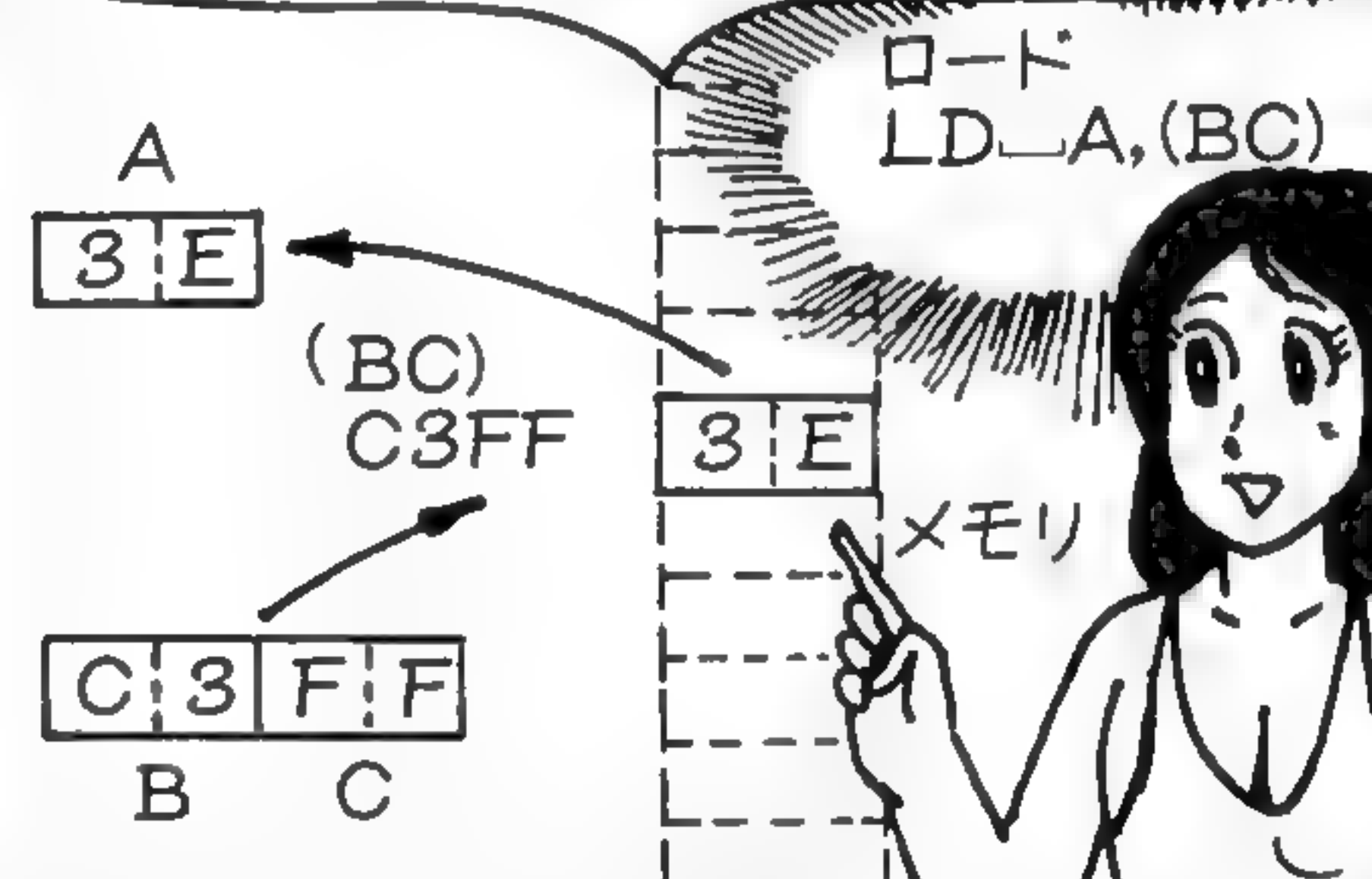
01 20 00

あれれ?

ご心配なく。マシン語はちゃんと01 00 20 です。

普通はリストを見やすくするために、ラベルとニーモニックは **TAB** キーを使ってずらせます。

エディタでソースプログラムをキーインしながら感じたことは、とても使いやすいニーモニックだなということです。たとえば8080の LDAX B よりもZ80の LD A,(BC)の方がラクラクとキーインできます。



CPUは8080や8085を使っているもニーモニックはこのZ80のものを使うことが多いようです。以上、簡単ですが、これでZ80ニーモニックのアセンブラの話はオシマイです。

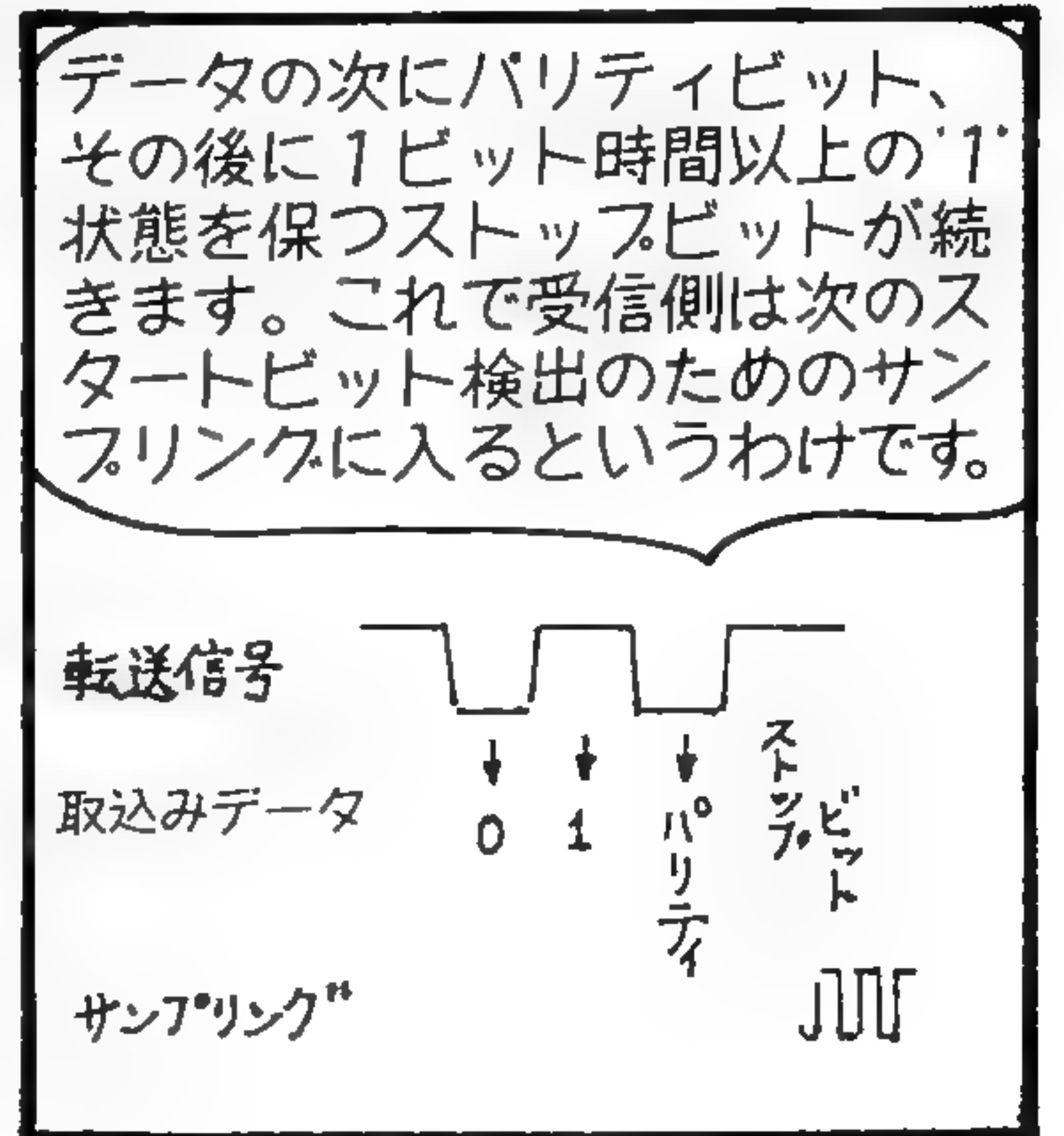
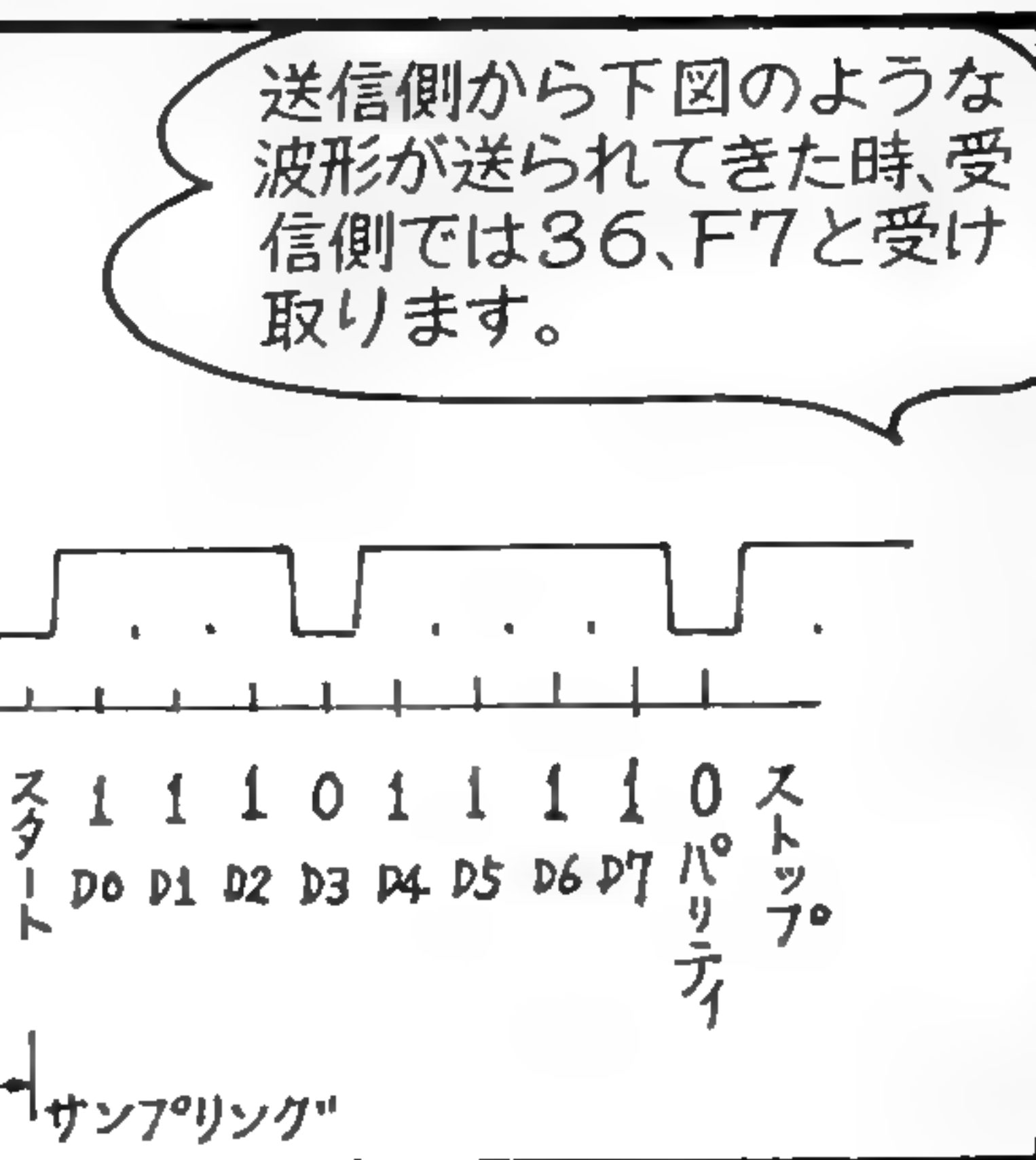
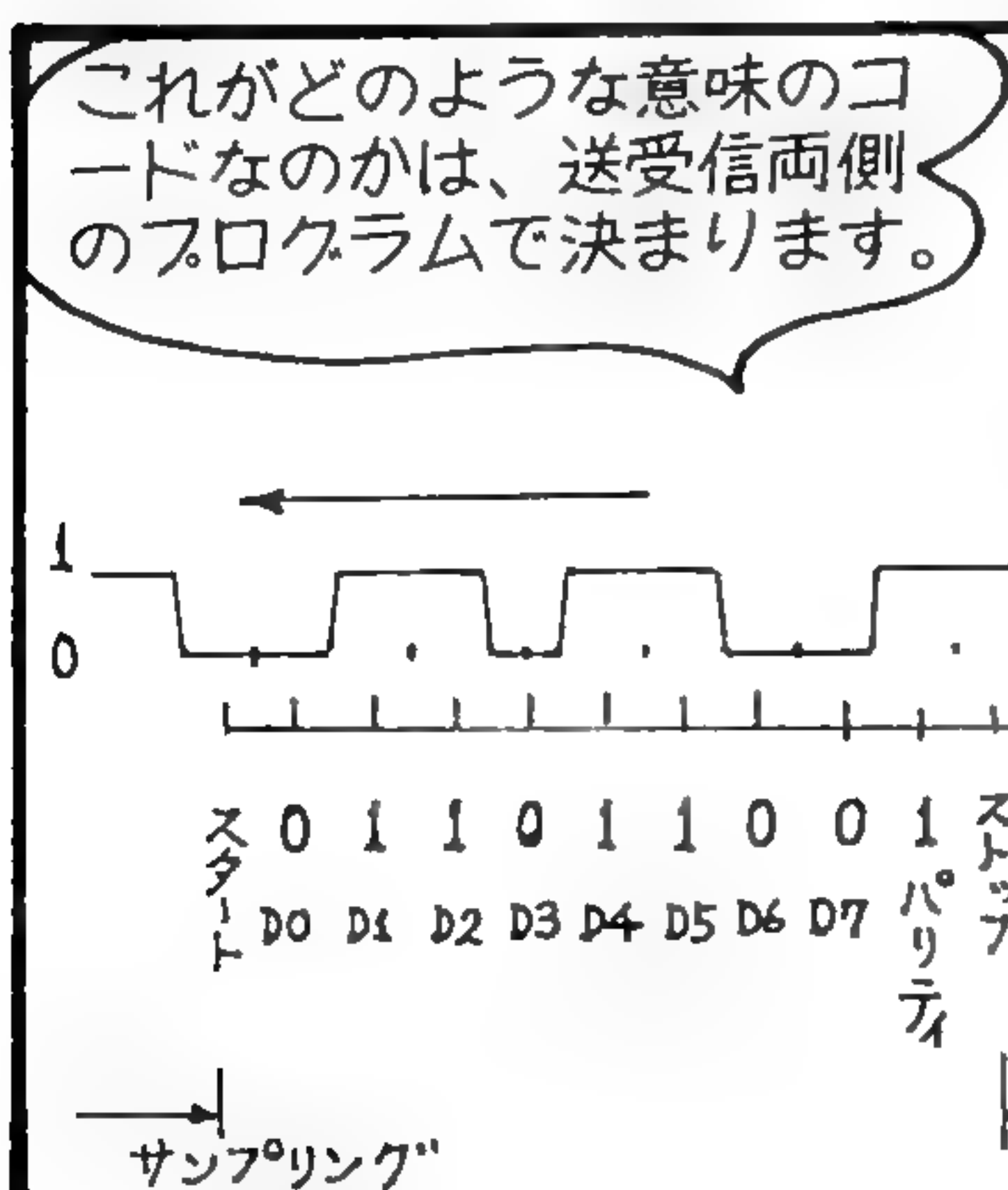
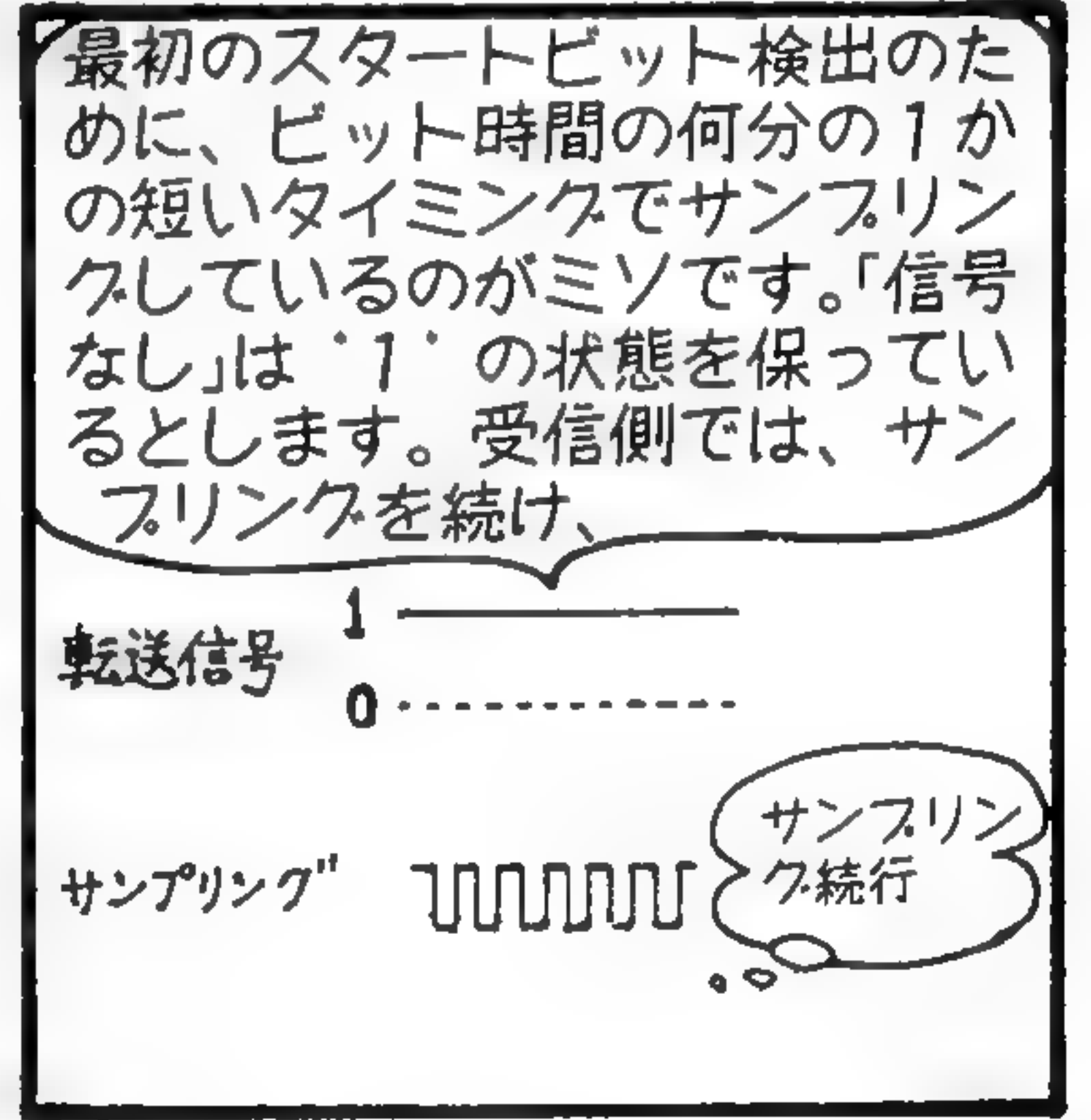
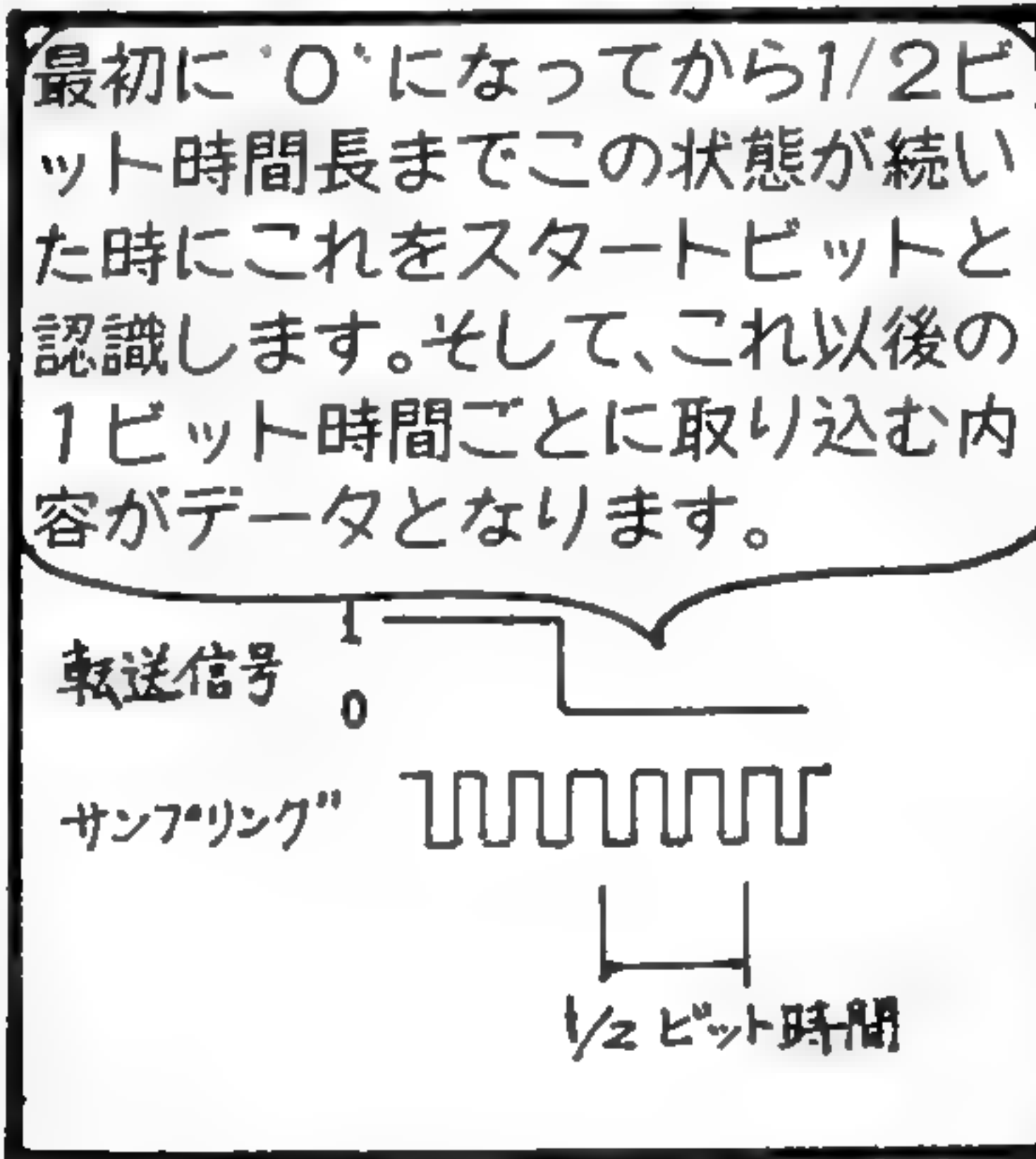
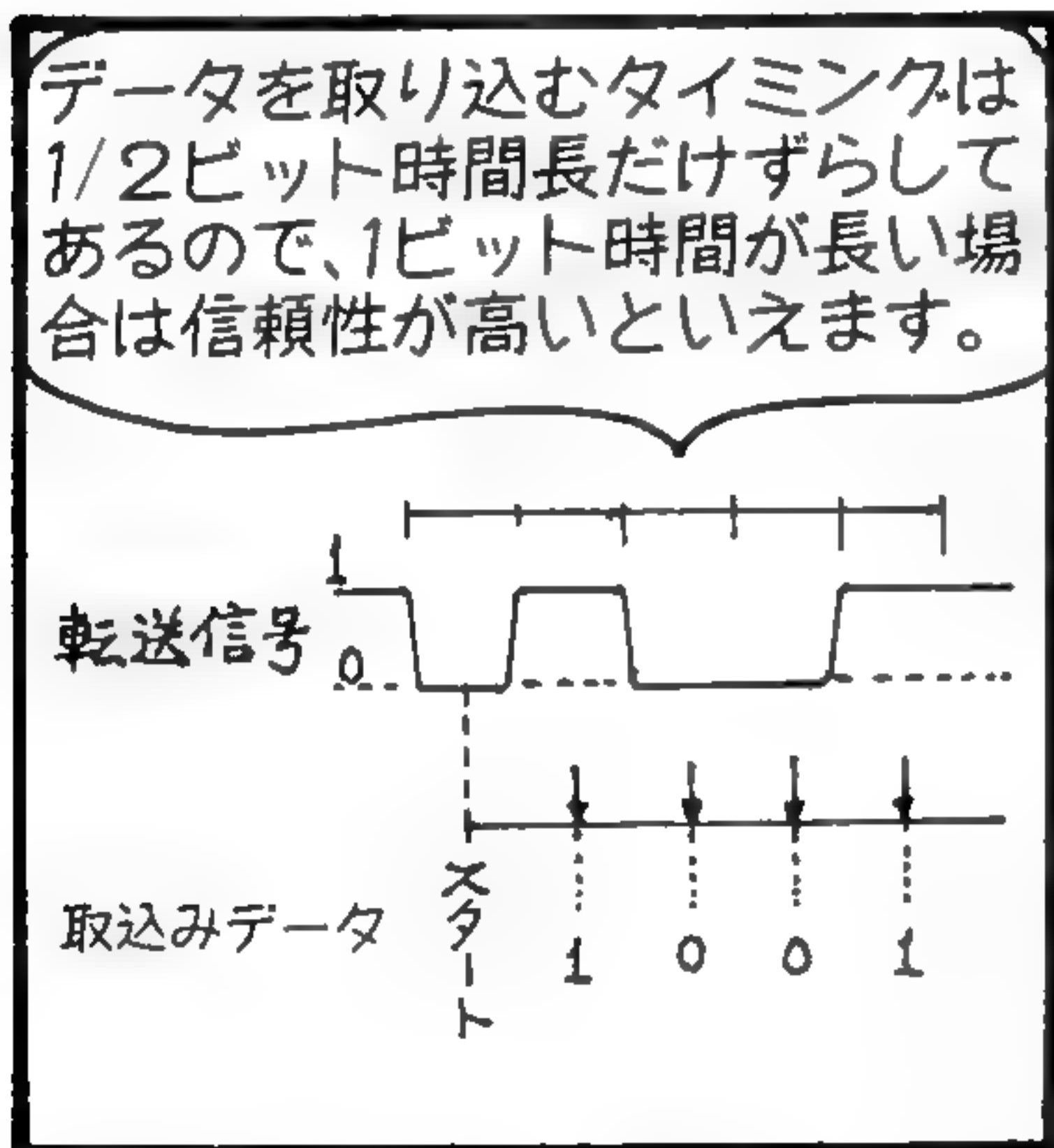
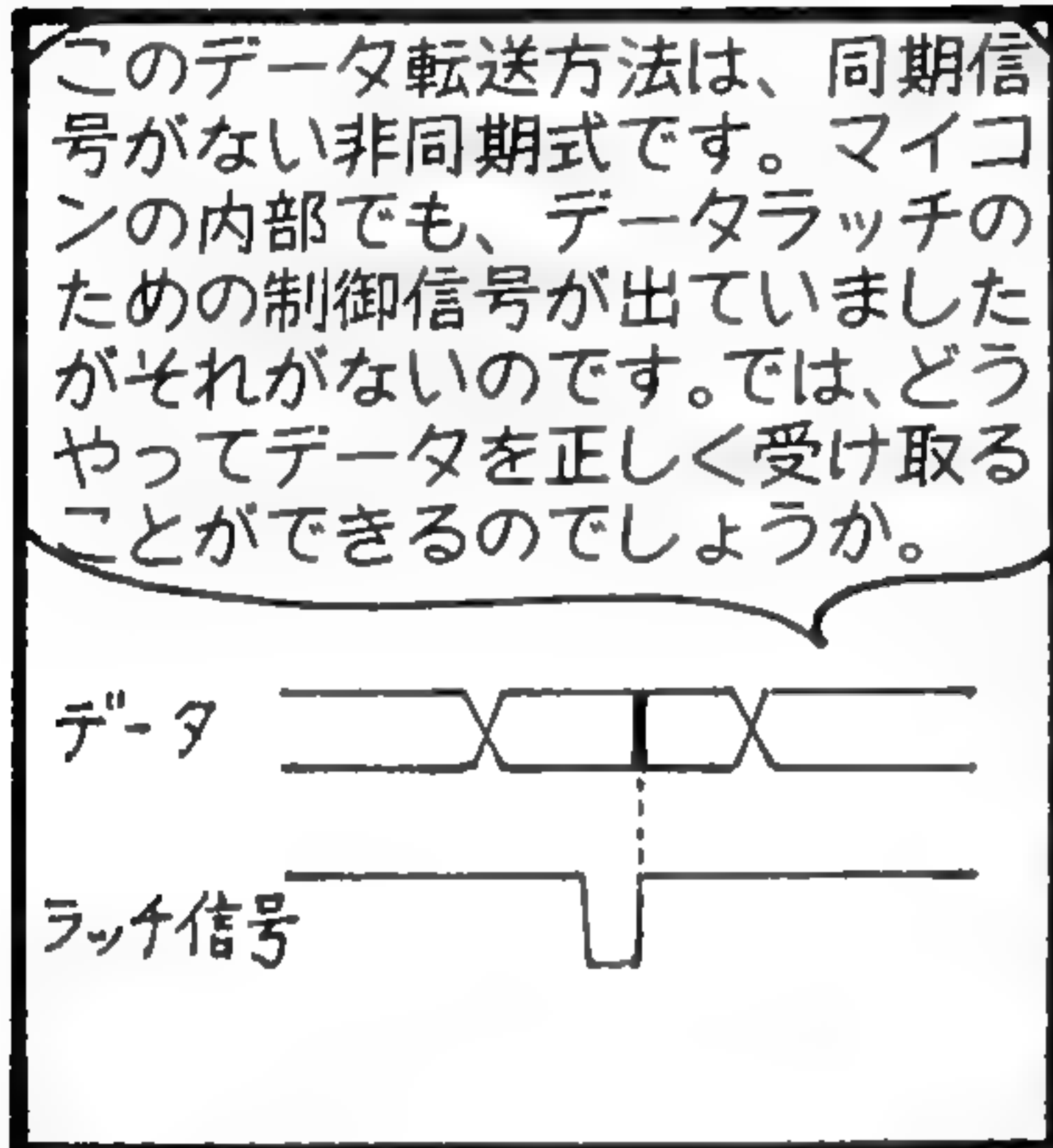








# ①RS-232C活用のハード

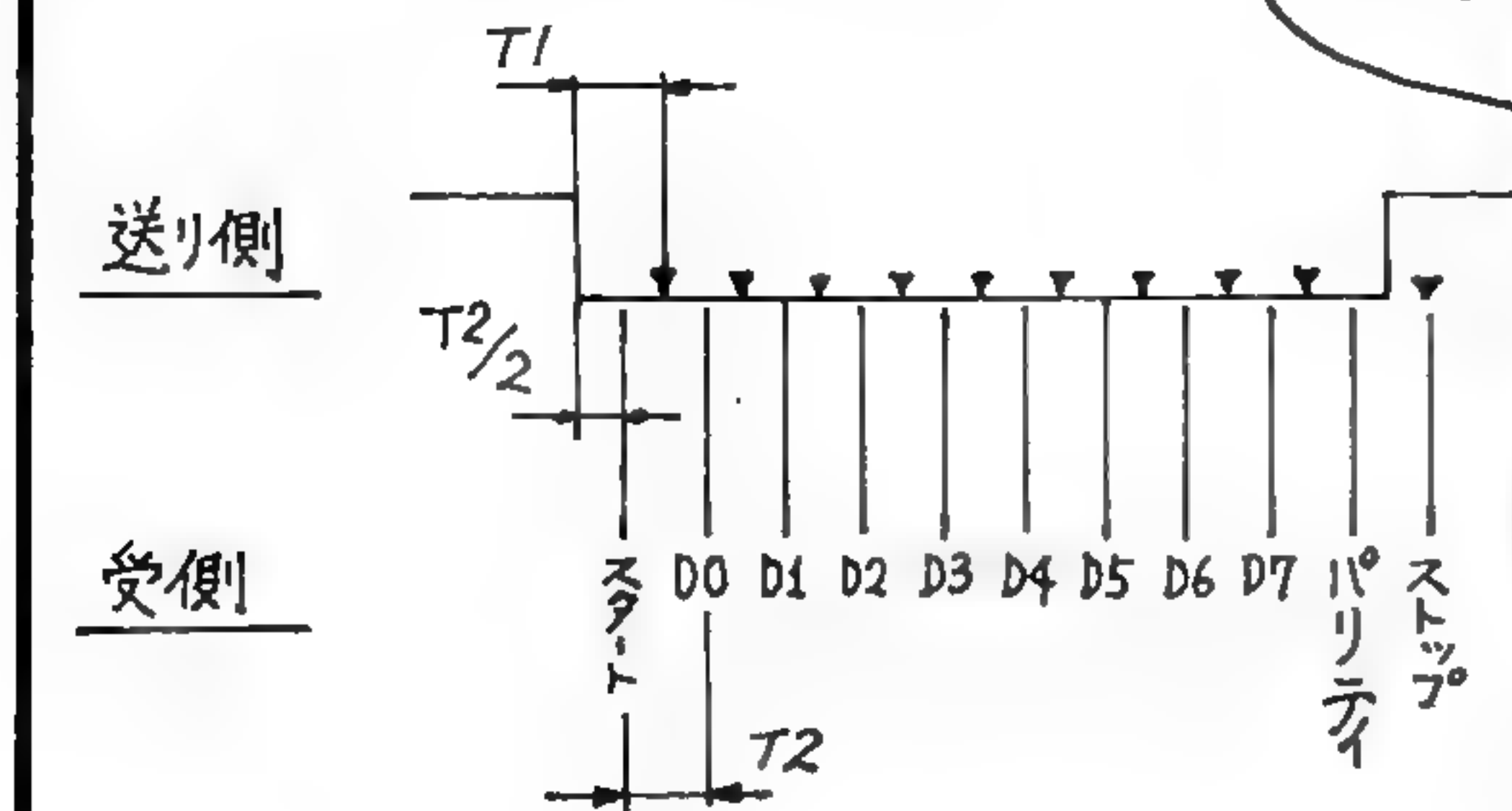




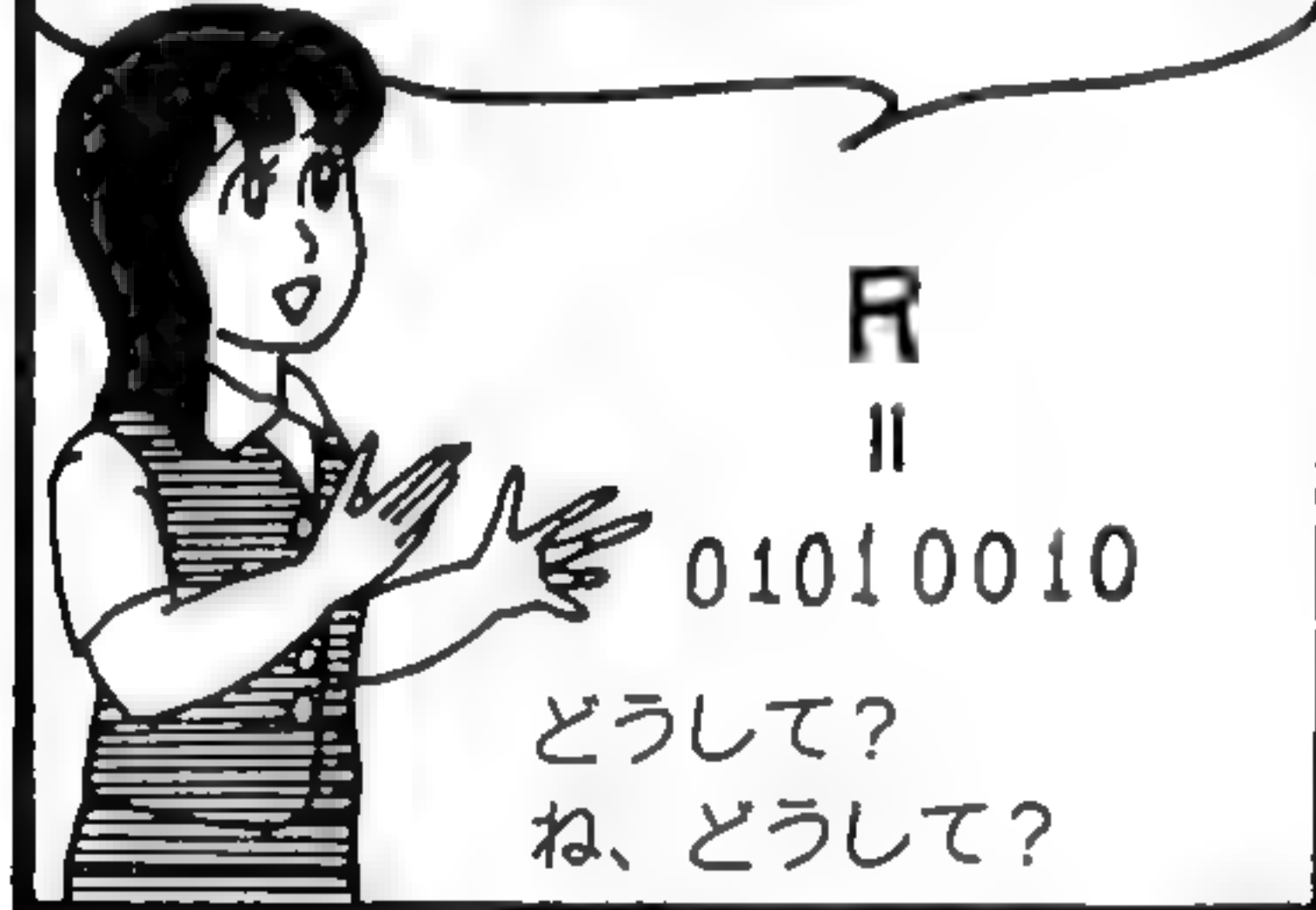
そこで、このデータ転送方式は調歩同期式と呼ばれます。

送信側と受信側の歩調が合っていないとダメということです。

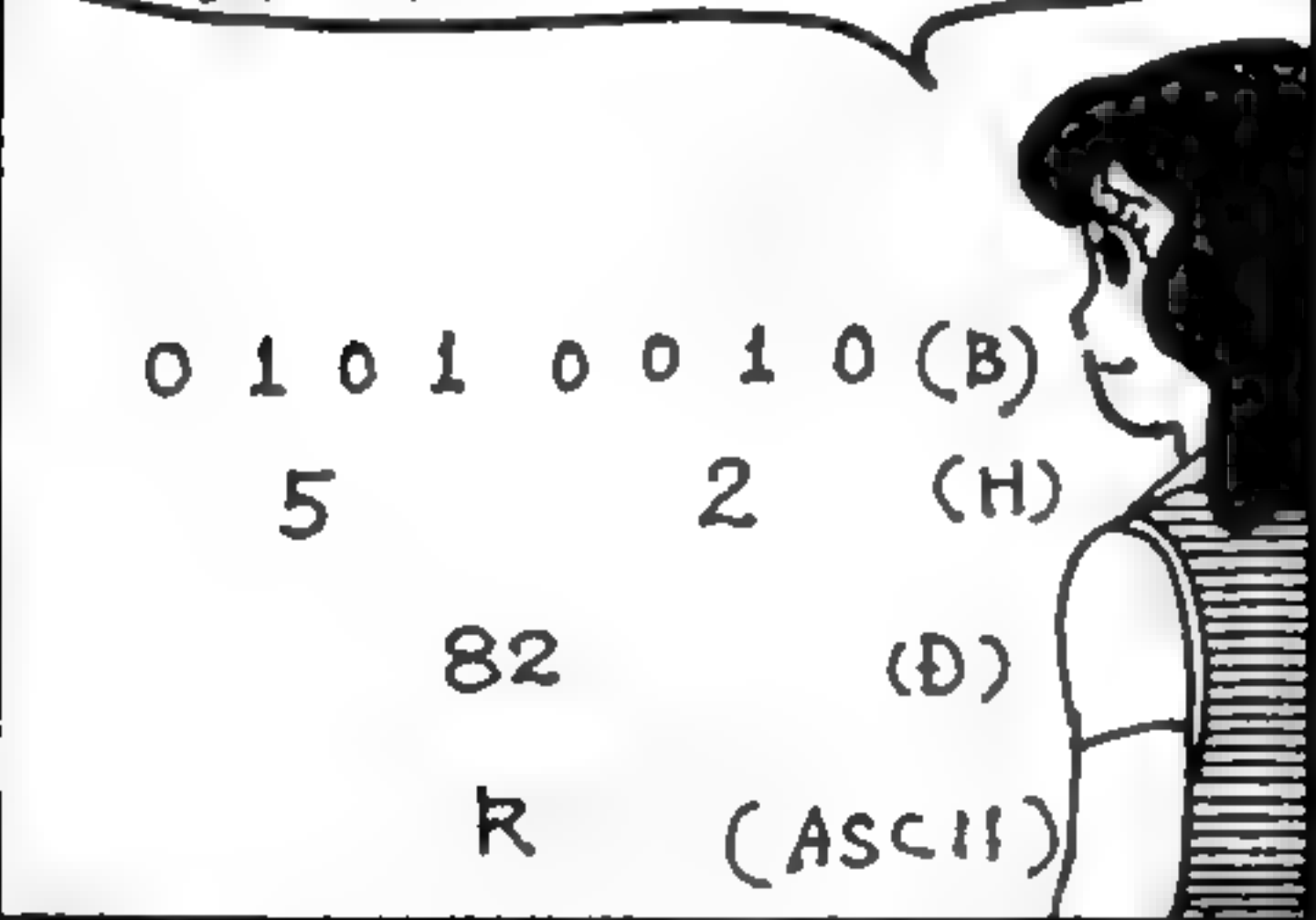
最初のスタートビットの検出だけでは、こうまきはいきません。1ビット時間、1コードのデータ長(ビット数)、パリティの有無、ストップビット数、などの条件を合わせておかねばならないのです。



たとえばRはアスキーコードで、10進表現では82、16進では52、2進では01010010となります。ポートから出ていくのは2進の電圧レベルの信号です。



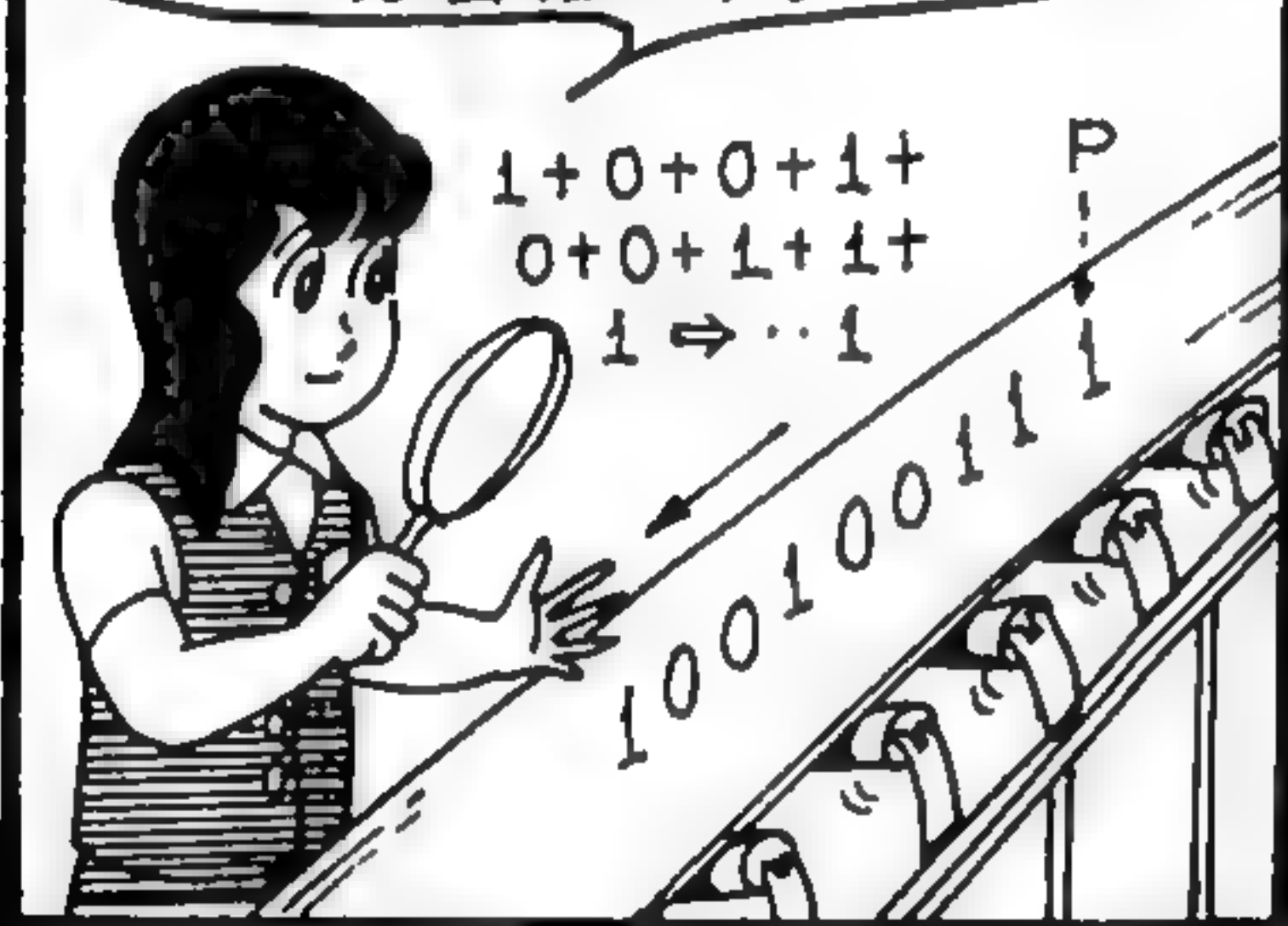
次に、データのコードを何ビットで表現するかを決めます。ふつうアスキー(ASCII)コードであれば7ビット、これにカナの入るJISコードであれば8ビット必要です。



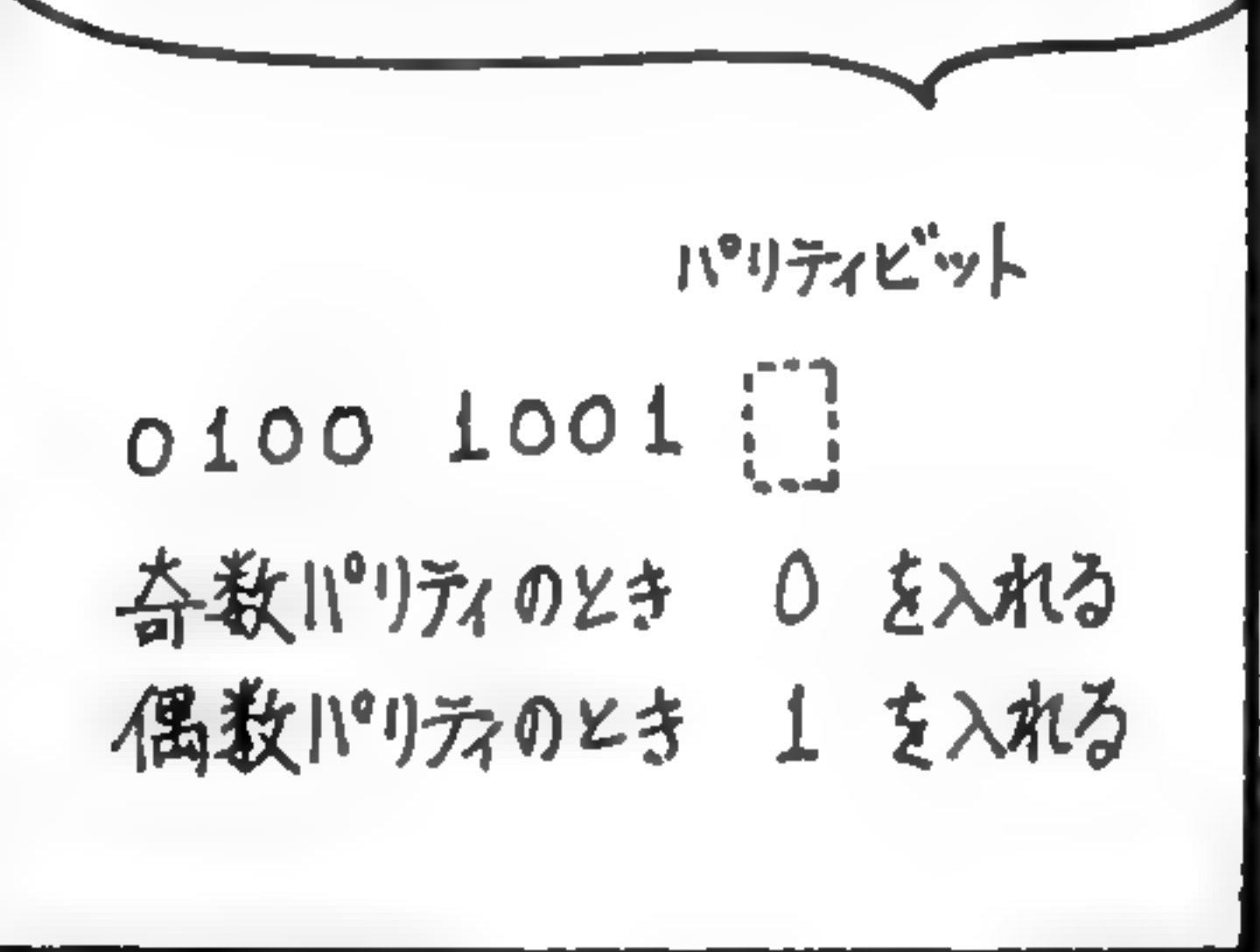
まず送り側のビット時間T1と受信側のデータ取込み時間間隔T2が等しくなくてはなりません。この単位はボー(ビット/秒)で示されます。



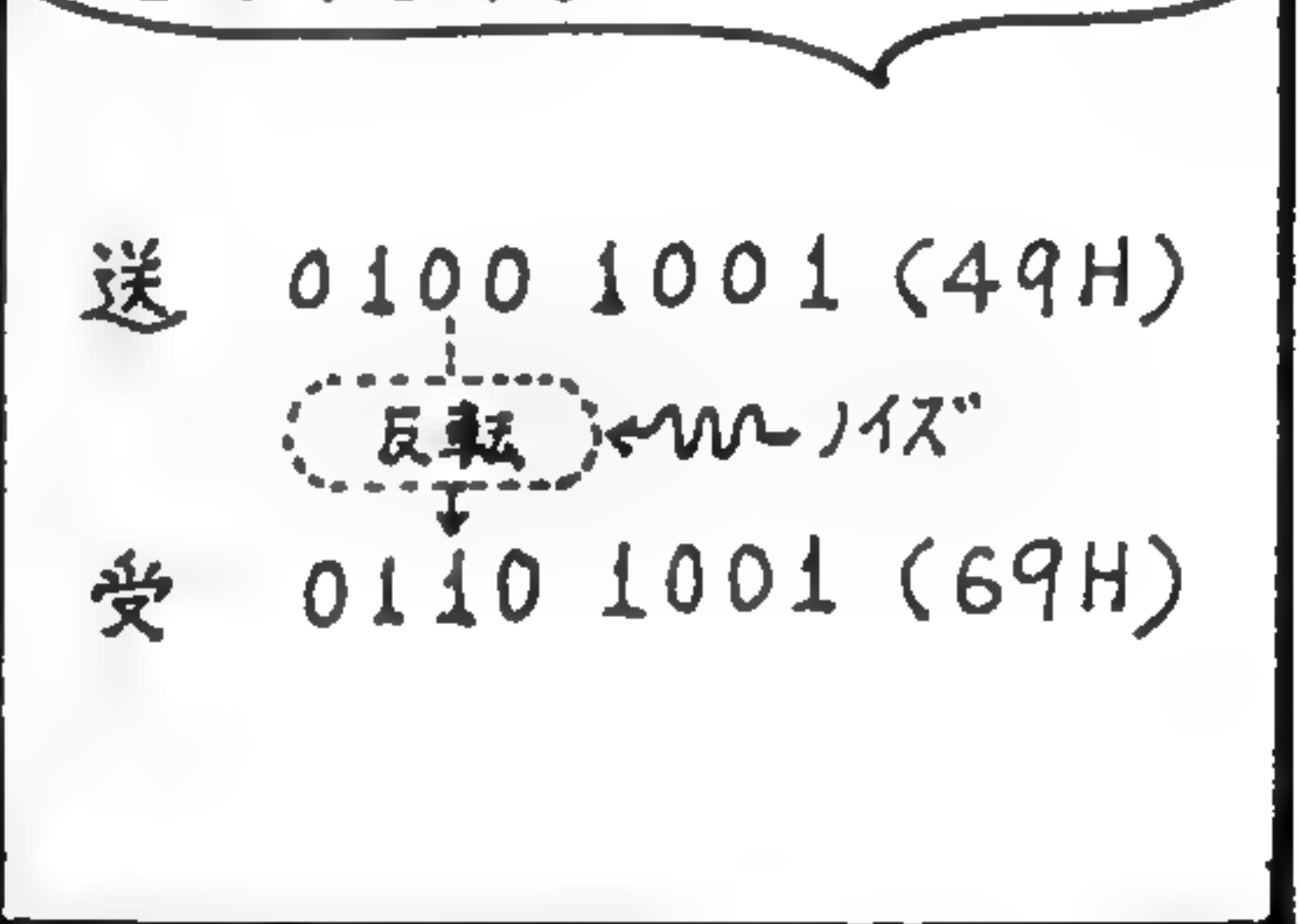
受信側でこれを奇数パリティとして受け取った時、各ビットをたし算し、パリティビットまでの合計の最小桁が常に1であれば合格です。



奇数パリティというのはデータ8ビットの中に含まれる1とパリティビットの合計が奇数になるようにパリティビットを0か1にセットするものです。



その次のパリティはビット検査方法のひとつで奇数パリティと偶数パリティの2種類あります。パリティビットをつけない場合もあります。



もし、アスキー城を埋められたら、すぐにも落城するだろう。

RS-232Cを自在に使いこなすには、まずアスキーコードの正体をはっきりつかんでおくことが必要だと思います。

ストップビットも1、2、3のいずれかをセットしておきます。調歩同期式ではこれらを同じ条件にセットしておかねばなりません。





## ②アスキーコードがカギ

CPUのマシン語とゴチャまぜにしないでくださいネ。CRT上の表示やキーボードの表示はキャラクタ、それを7ビットのコードに割り当てたものがアスキーコードです。

CPUの  
マシン語

全然別  
のもの

アスキー  
コード

7ビットでの組み合わせは128通りあり、アスキーで使う文字や記号はこれで足りていました。

	0	1	2	3	4	5	6	7
0	128種の文字を割り当てることができる。							
1								
2								
3								
...								
...								
...								
...								
...								
...								
...								
...								
...								
...								
...								
D E F								

そこで私たちの使う文字記号とCPUが使う時に必要なコードとの間に関連を持たせなければなりません。このコード変換で一番広く普及しているのがアスキー(ASCII)コードです。

アスキーコード	文字
0100 0001	A
0100 0010	B
0100 0011	C
0100 0100	D

私たちが使うのは、0~9までの数字、A~Zまでのアルファベット、それに+、-、(、\*、=などの特殊文字です。(アメリカの人はこれで足りるわけです)

ABCDEFGHIJKLMNOPQRSTUVWXYZ  
abcdefghijklmnopqrstuvwxyz  
01234567890

8ビットでは7ビットよりもさらに128多くコードを割り当てることができます。

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	ASCII コード  (128)								カタカナ コード  (128)							
1																
2																
3																
...																
...																
...																
...																
...																
...																
...																
...																
...																
...																
...																
C D E F																

カタカナを入れた8ビットコードは「アスキーコード」ではなく「JISコード」になります。しかし7ビット分はアスキーに準じたコードにしてあります。この部分もJISで独自に決めたのではコードに国際的な互換性がなくなり不利になるのは日本です。

ASCII    カタカナ  
7ビット    8ビット

ところが日本人の場合は、ひらがな、漢字、カタカナも合わせて使いますので128のアスキーコードだけでは不便です。そこで、せめてカタカナだけでもということで、8ビットにし、増えた128コードにカタカナを割り当てたのです。

I LOVE JAPAN  
ニッポン ダイスキ

電子回路側からのアウトプットの際はアスキーコードをデコードし、文字記号に戻さねばなりません。

デコード

01010010    アスキーコード

R

電子回路でのデータの取扱は、文字記号の場合このアスキーコードで行なっているわけです。

01010010

このデータがマシンコードなのか数値なのかアスキーコードなのかはプログラムしだい。

メモリ    アドレス

インプットの際はこの文字記号をエンコードしてアスキーコードにします。

キーボード    文字

エンコード

01010010    アスキーコード



そこで説明書などには16進表現で書いてあります。

Rのコードは52です。

	0	1	...	5	...	F
0						
1						
2						
...						
F						

↓

R

人間同士でアスキーコードの話をする時や説明をしたい時にはコードそのものでは非常に不便です。

01010010

?

このようにアスキーコードというのは7ビットで表現されるコードに文字や記号を割り当てたものですが、

001Fまでの32通りのコードには文字記号が割り当ててありません。ここには制御用コードが必要に応じて割り当てられるようになっています。

	0	1	
0	文字・記号は	割り当てられていない	
1			
2			
3			
4			
5			

何番目に割り当てられた文字かを知りたい時は10進表現の方がよくわかりますが、実用性はどうでしょう。

&H52は0から数えて83番目、数字では82ということになります。

BASICでは16進表現であることを示すために&Hをつけます。ポケコンは&だけでした。

&52      &H52



たとえば10進数の5は35というコードになり、このコードを10進数で表現すると53になります。

どうしてこれが同じなの?

53 (アスキー10進)

5

数字は30から39に0から9が割り当てられています。

	3
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
A	:

次の20から文字記号が割り当てられています。20は空白コードです。20をデコードしたら「(空白)」になります。

	2
0	空白
1	!
2	"
3	#
4	\$
5	%
6	&

アスキーコードで文字記号を見た場合、数字<英大文字<英小文字の順になっています。

上桁 3ビット (0~7)

	0	1	2	3	4	5	6	7
制御コード								
特殊記号								
数字								
英大文字								
英小文字								
記号								

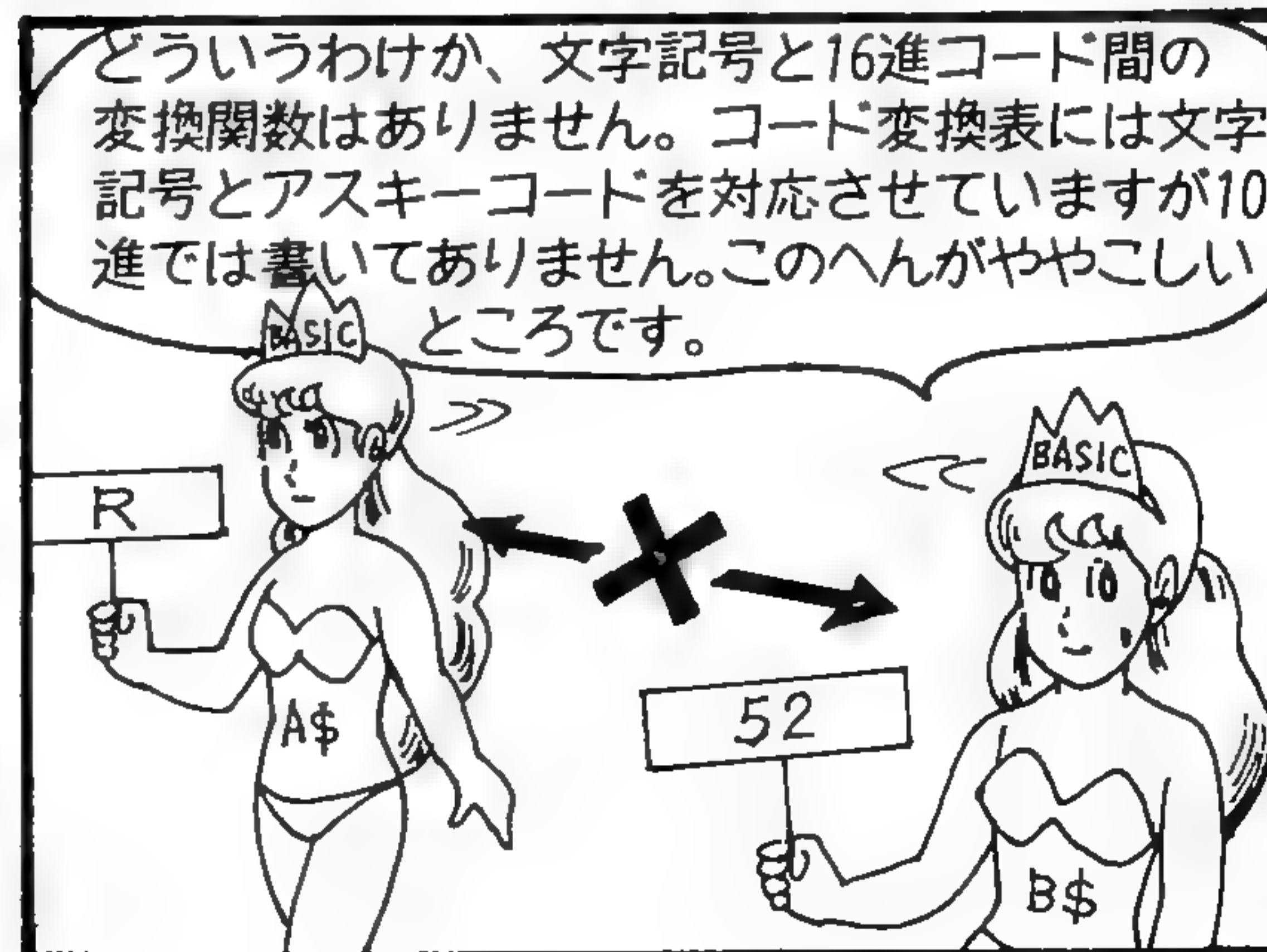
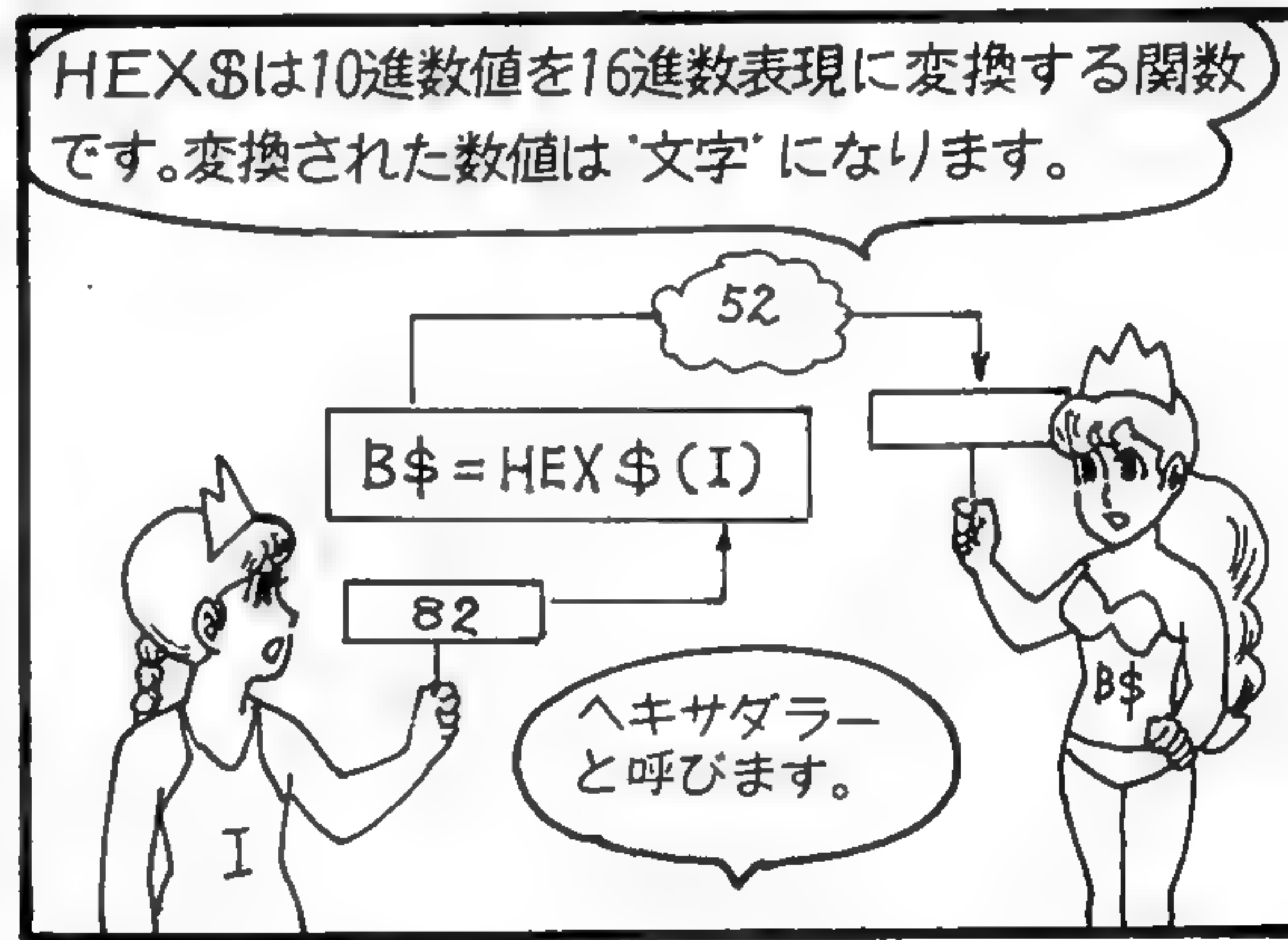
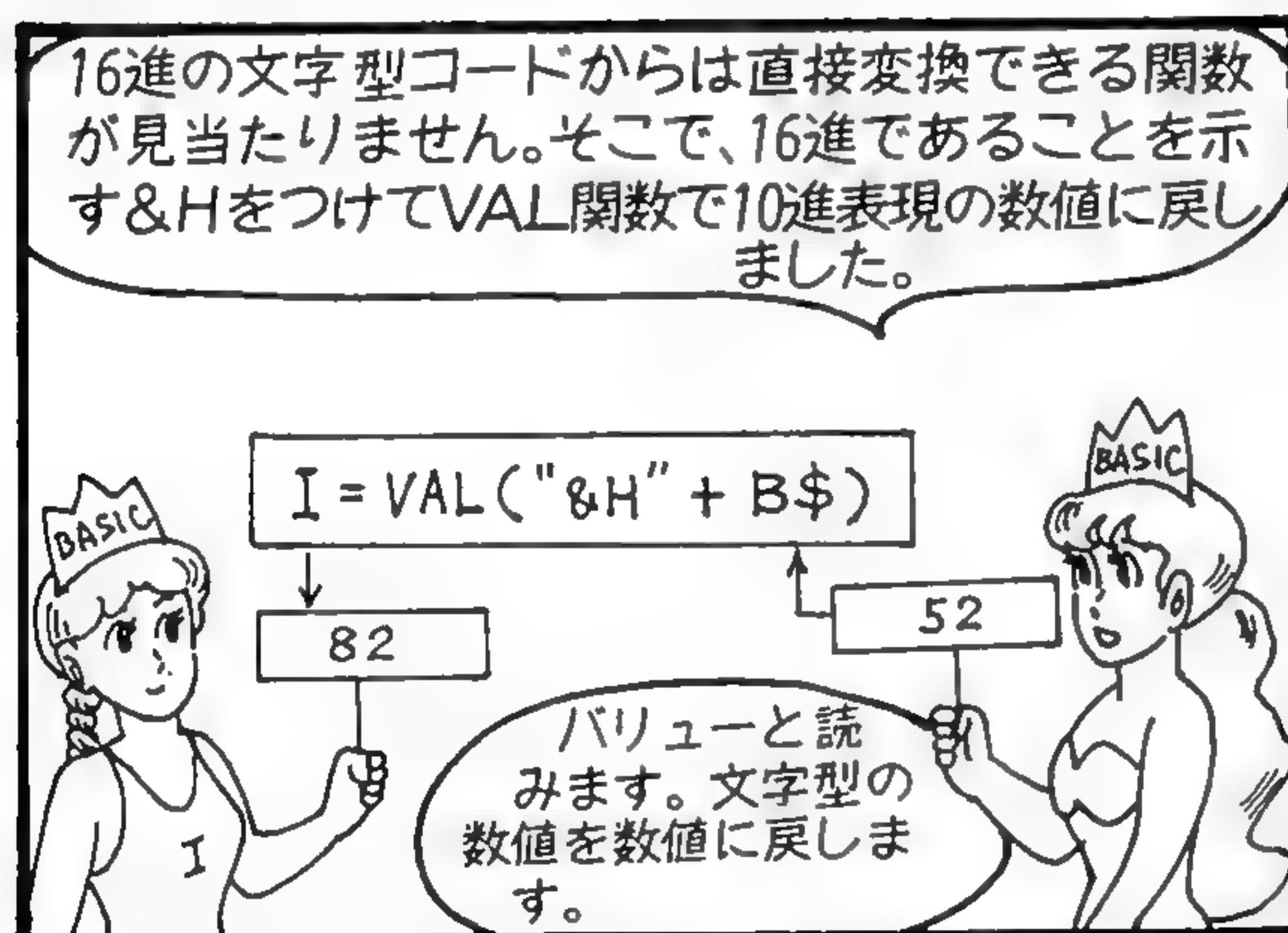
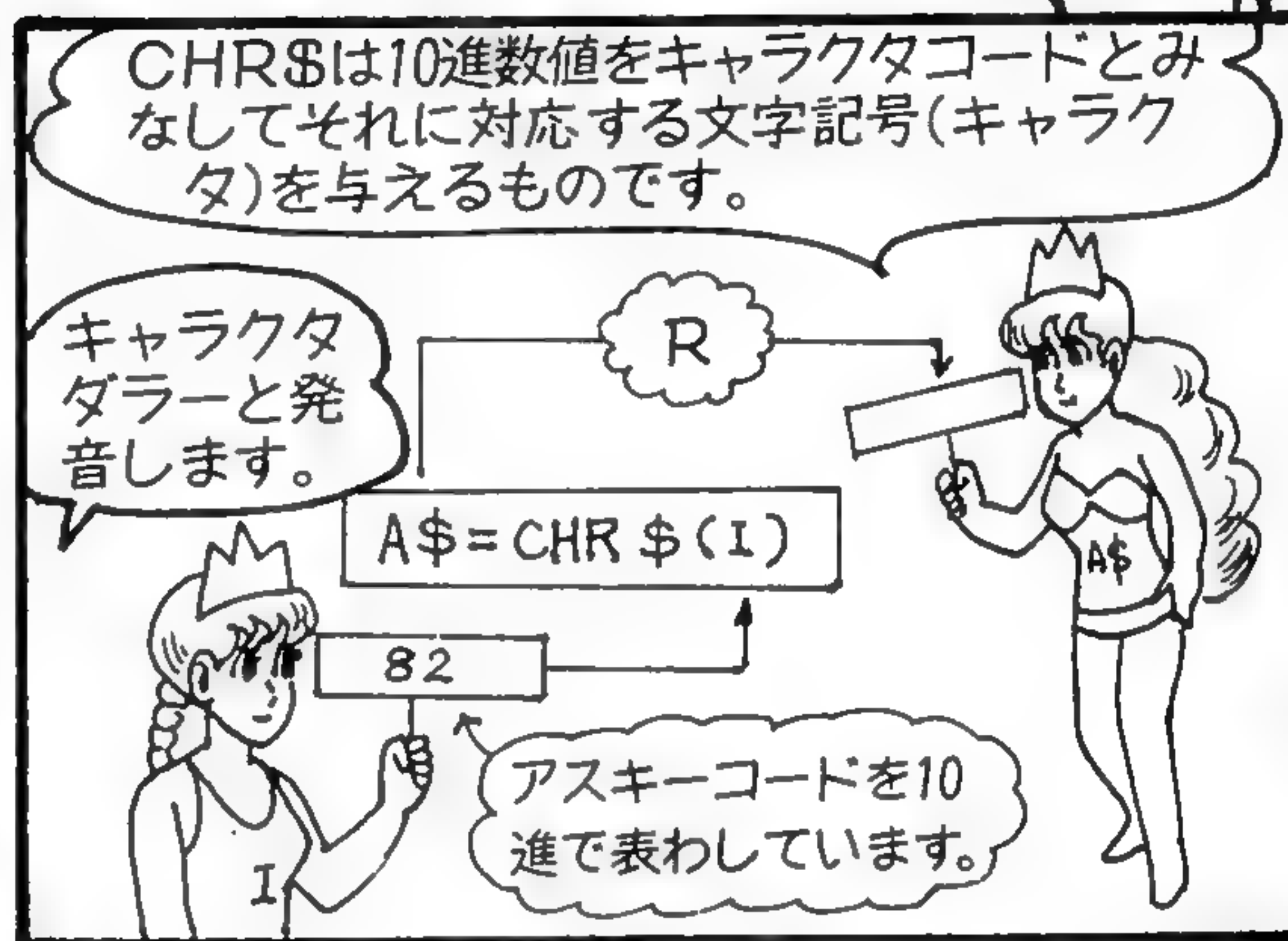
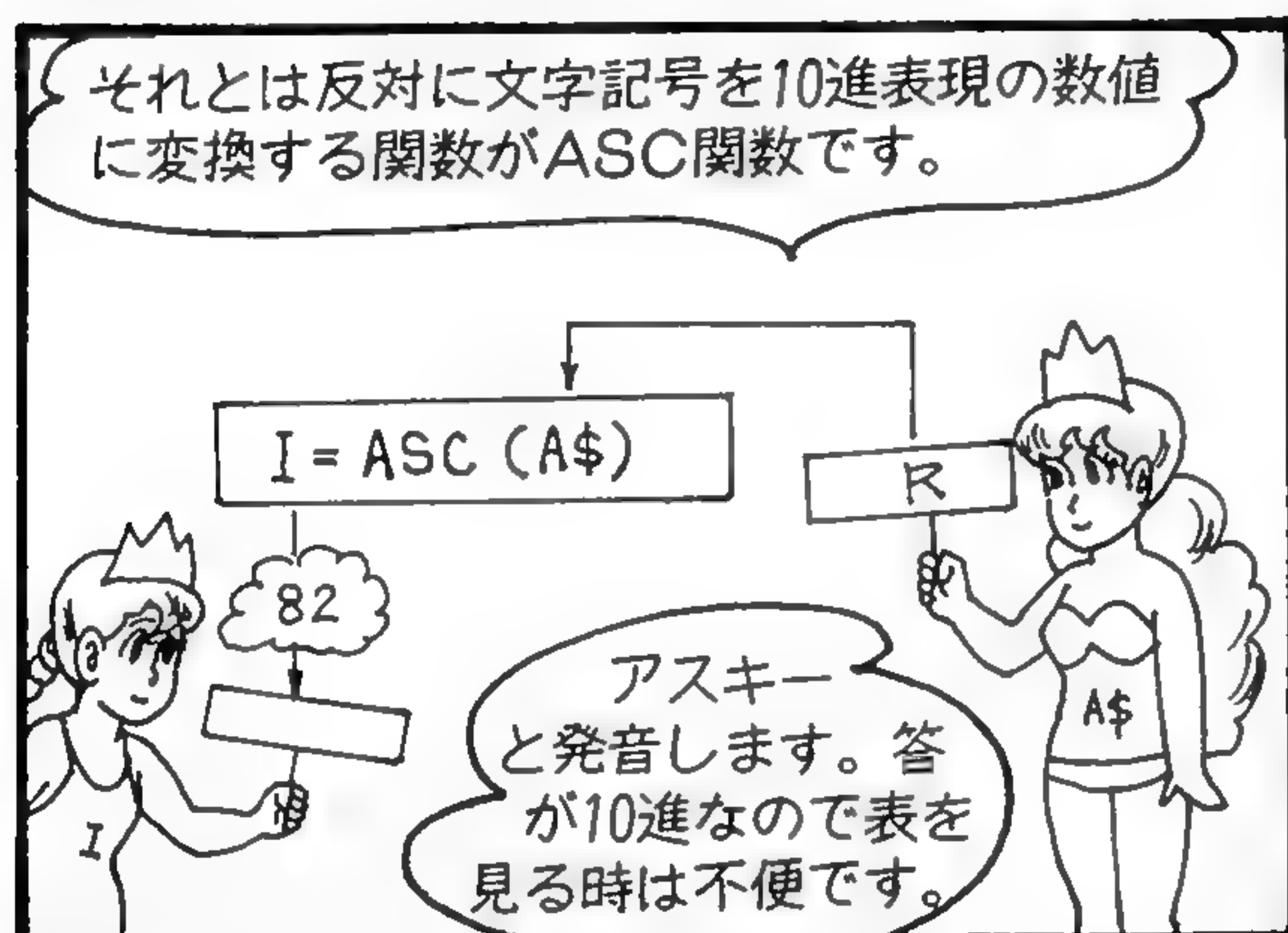
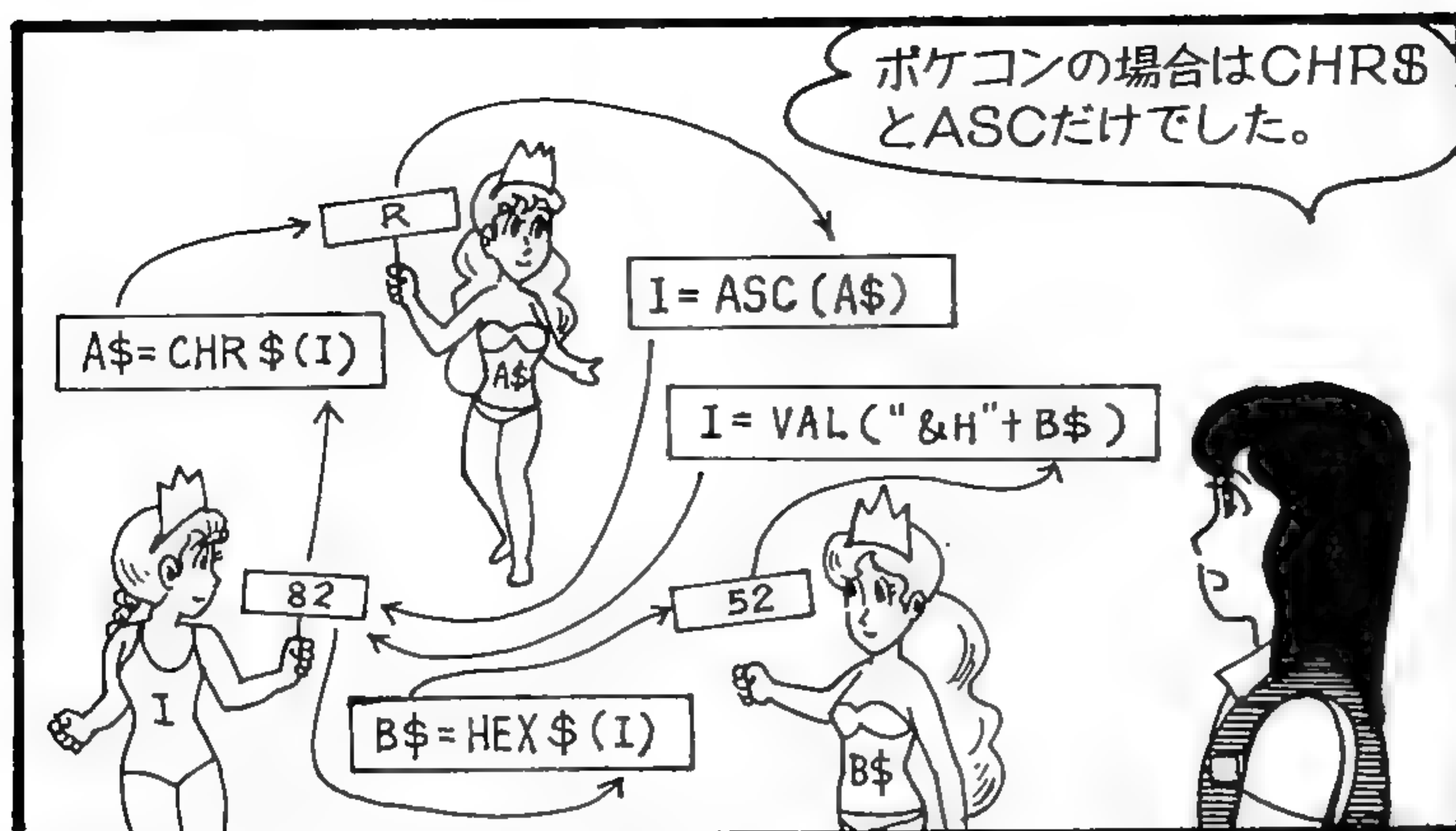
下桁 4ビット (0~F)

英大文字は41のAからはじまって5AのZまで、英小文字は61のaから7Aのzという具合になっています。

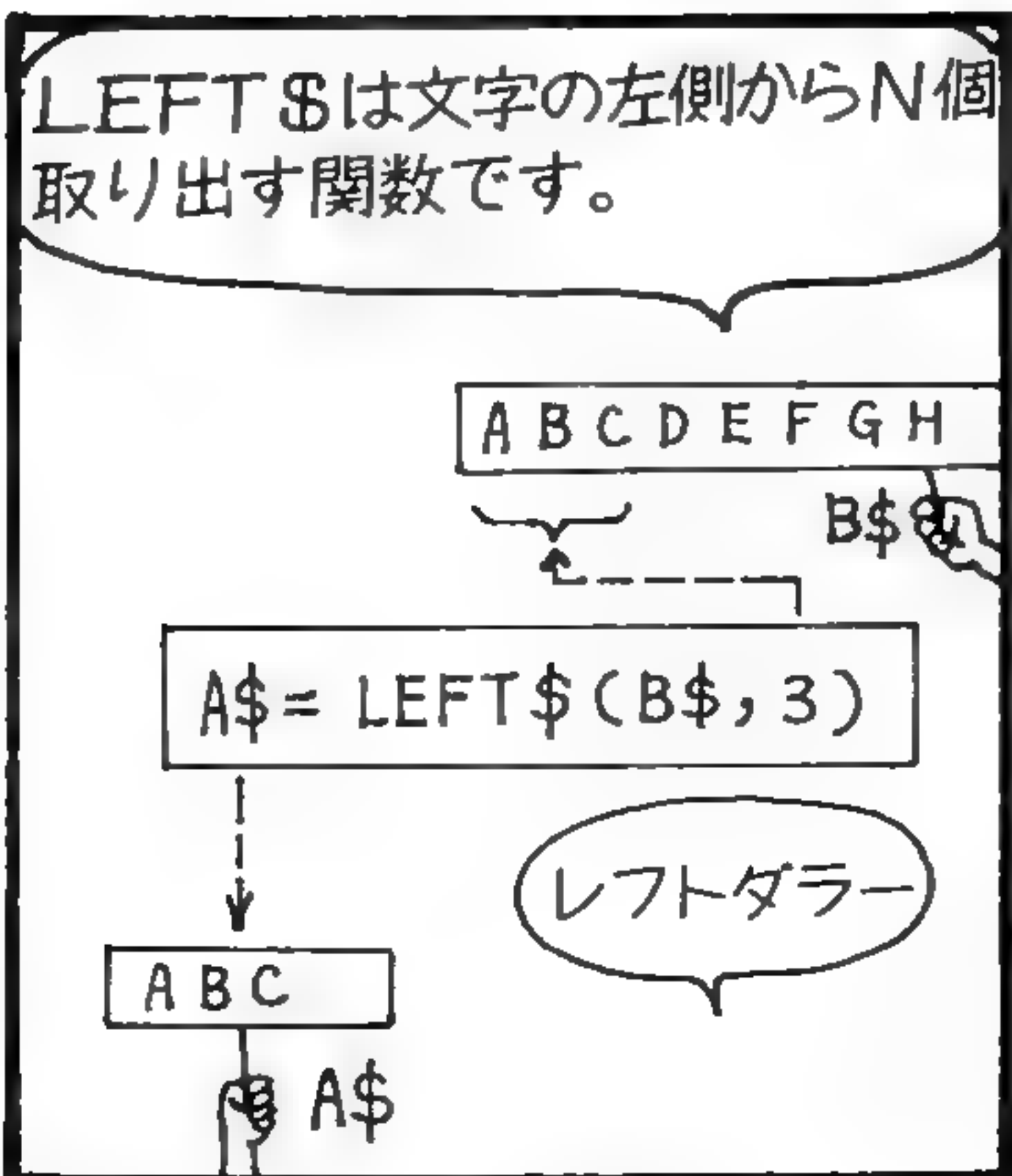
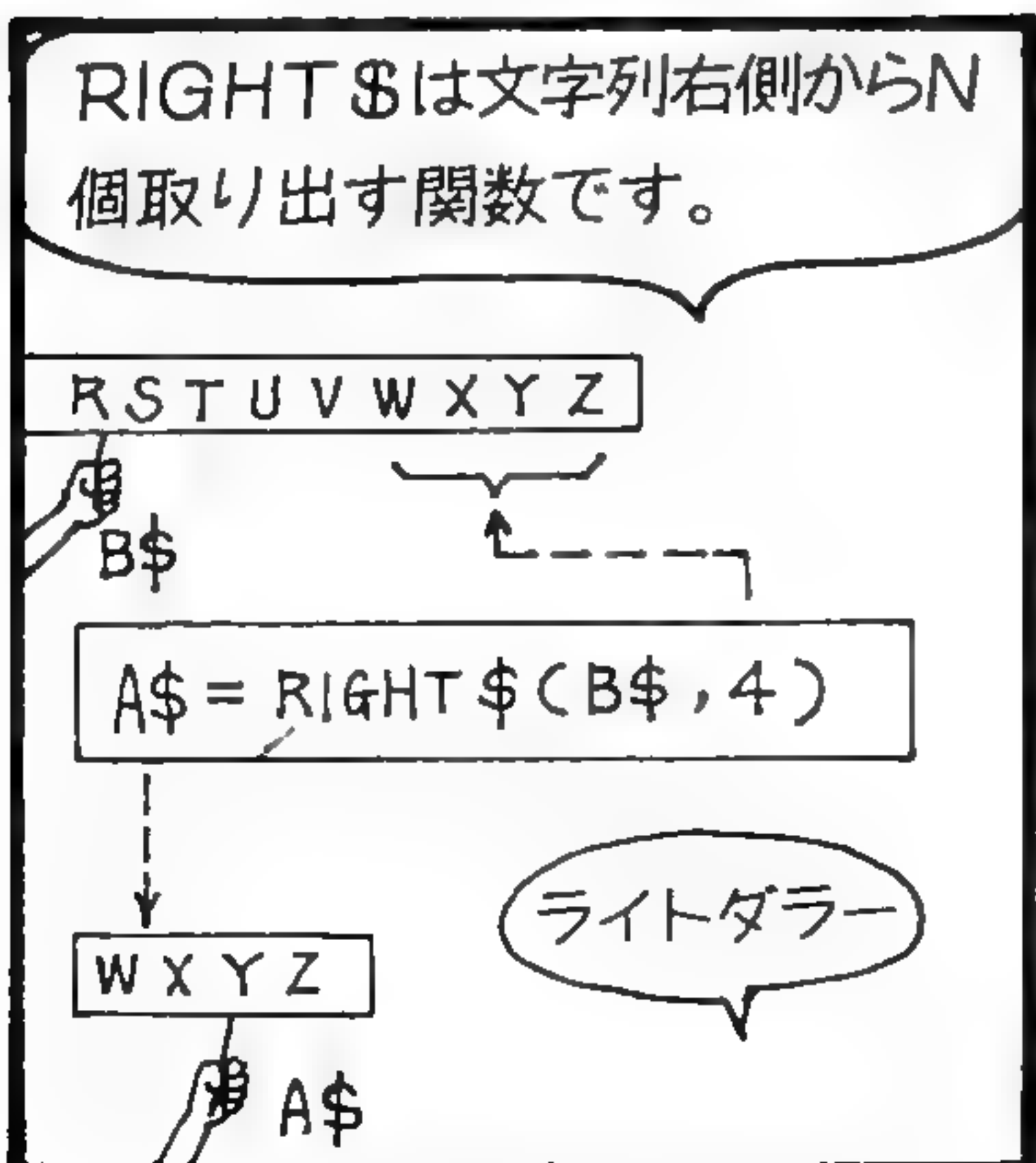
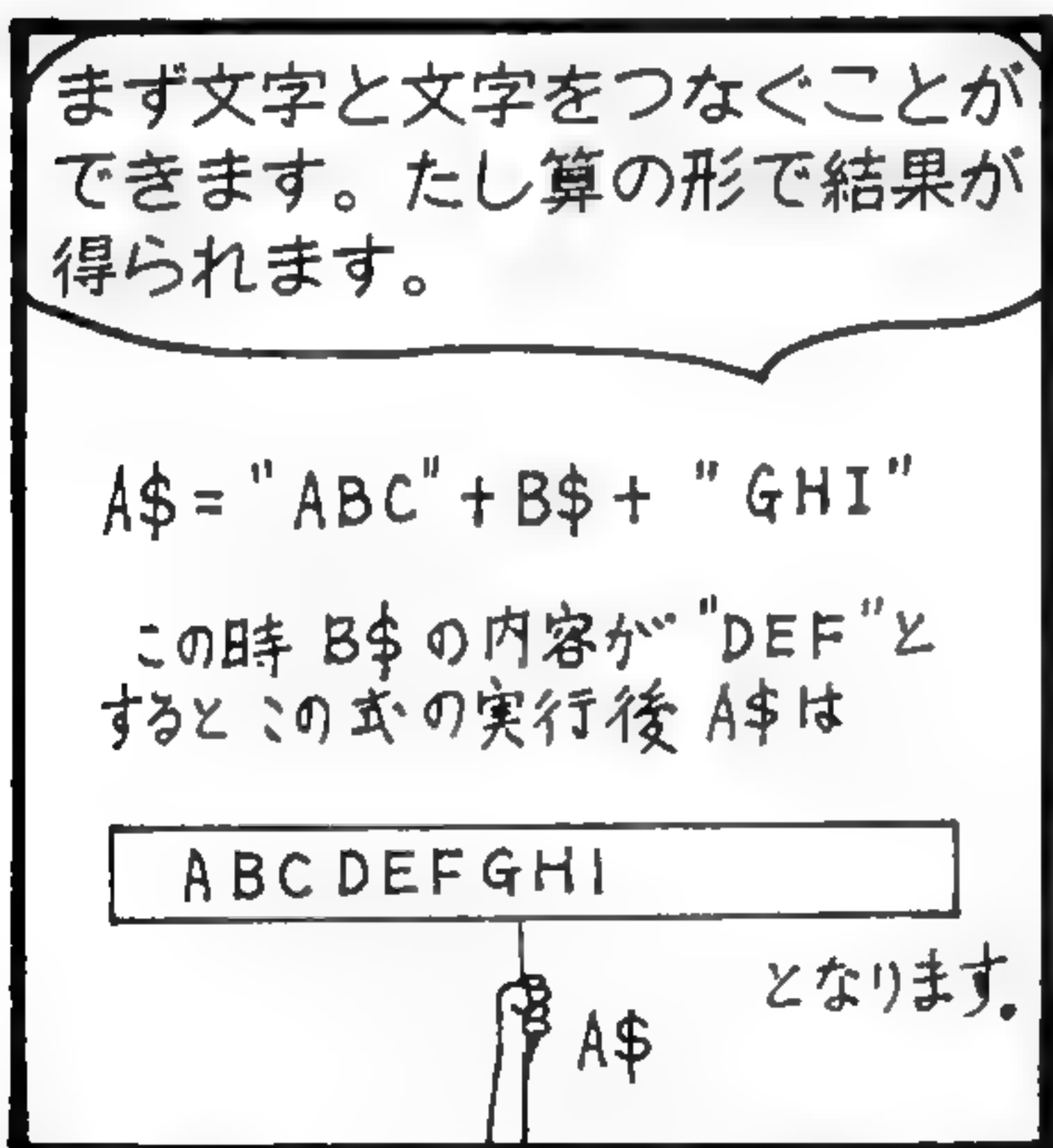
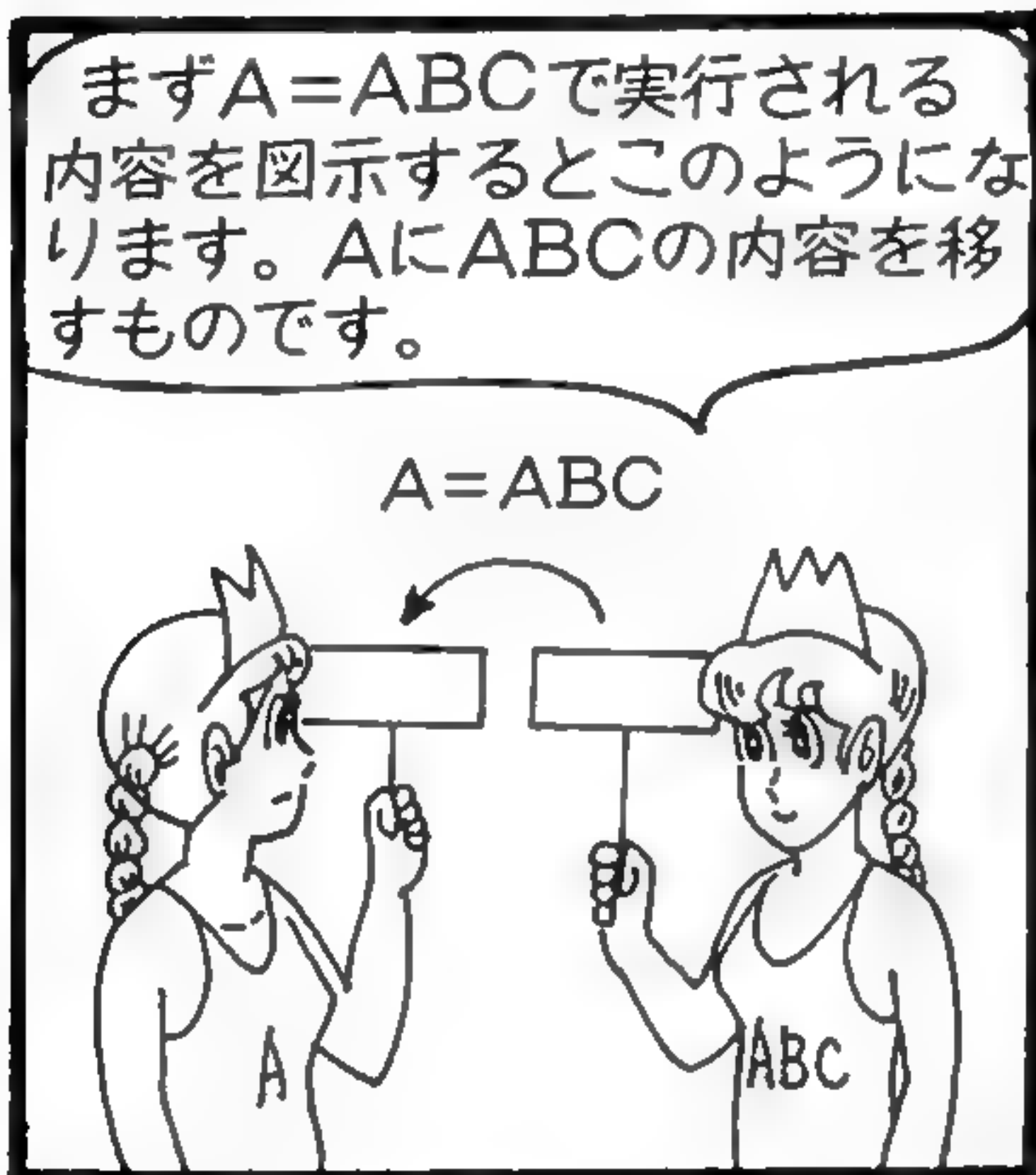
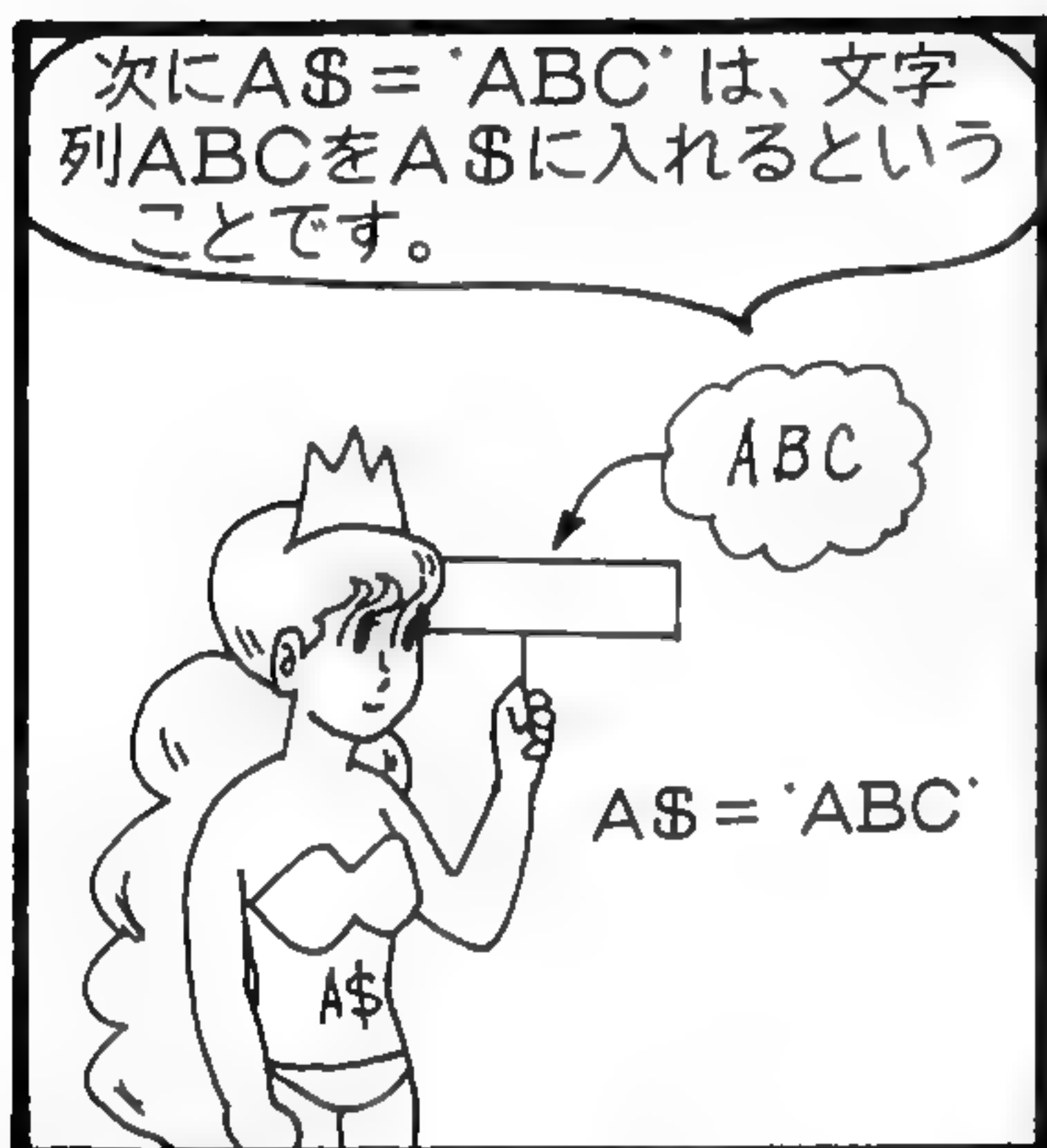
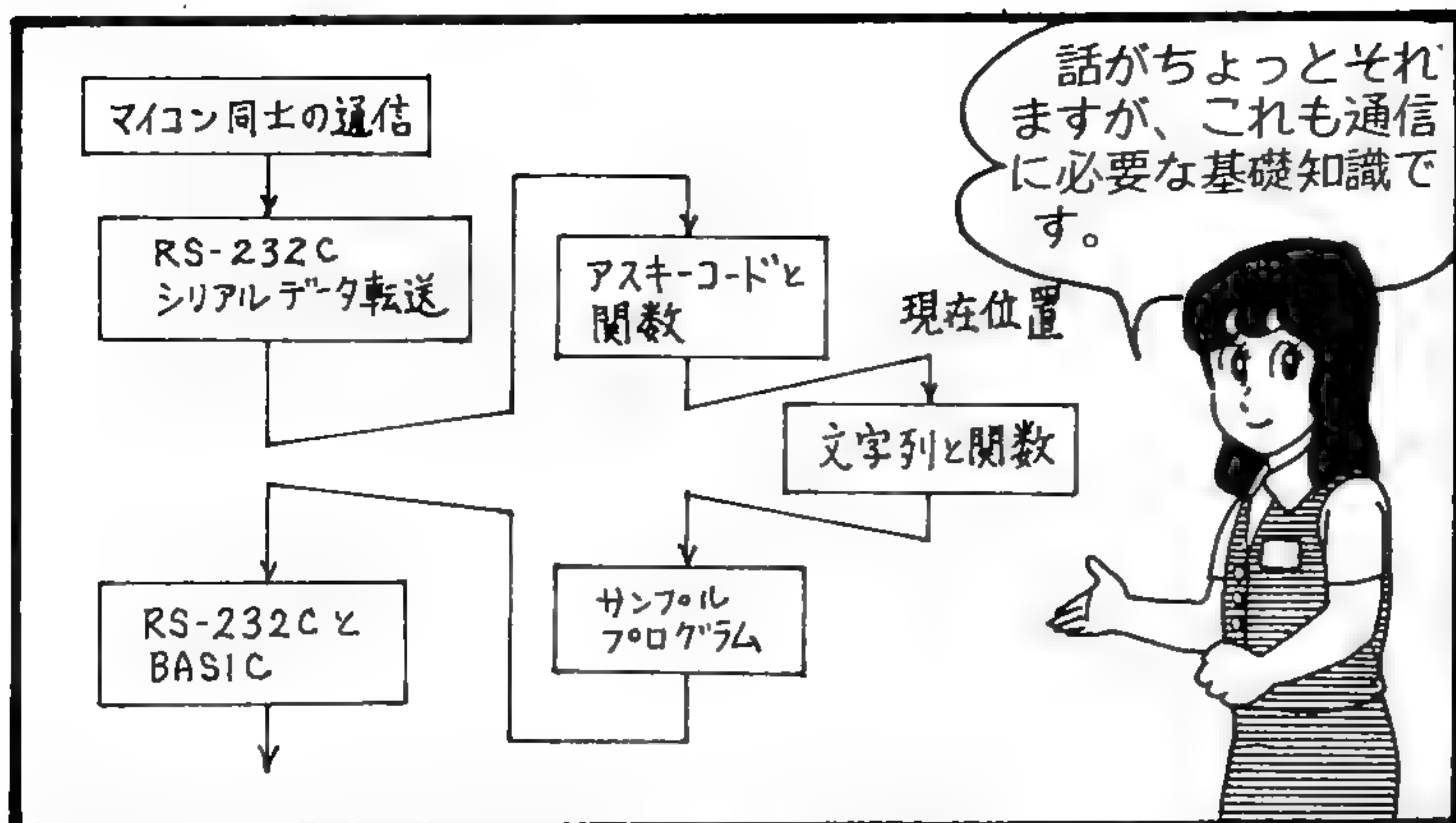
	4	5	6	7
0	@	P	,	p
1	A	Q	a	q
2	B	R	b	r
3	C	S	c	s
4	D	T	d	t
5	E	U	e	u
6	F	V	f	v
7	G	W	g	w



### ③ BASICで アスキーコード を操作









LENは文字列の長さを示す関数です。結果は数値になります。

ABCDEFGHI

B\$

A = LEN(B\$)

9

レンクス

A

MID\$関数はランダムファイルで重要な役割を果たしました。

MID\$は文字列の左側から数えてI番目の文字からN個取り出す関数です。

1 2 3

ABCDEFGHIJKLMN

B\$

A\$ = MID\$(B\$, 3, 4)

CDEF

A\$

ミッドダラー

IF800-30の場合は '&H' をつけると16進として0~9、A~Fを受け付けてくれました。

46

A\$

A = VAL("&H"+A\$)

71

46は文字、71は数値です。

A

VALはSTR\$の逆関数です。0~9のキャラクタから成る文字列を数値に変換する関数です。

3.14 G 5 7 8

A\$

A = VAL(A\$)

3.14

バリュー

A

STR\$は数値を文字列に変換する関数です。ランダムファイルで使ったMK\$関数とはタイプが違います。

12.34

A

A\$ = STR\$(A)

12.34

A\$

ストリンダラー

この秘密は文字列の各キャラクターが計算機内部ではアスキーコードになっていることにあります。

アスキーコードの  
数値表現

49

97

65

アスキー  
コード '&H'

31

61

41

キャラクタ  
コード

1

a

A

また、いくつかの文字列をキャラクタ順に並べることができます。

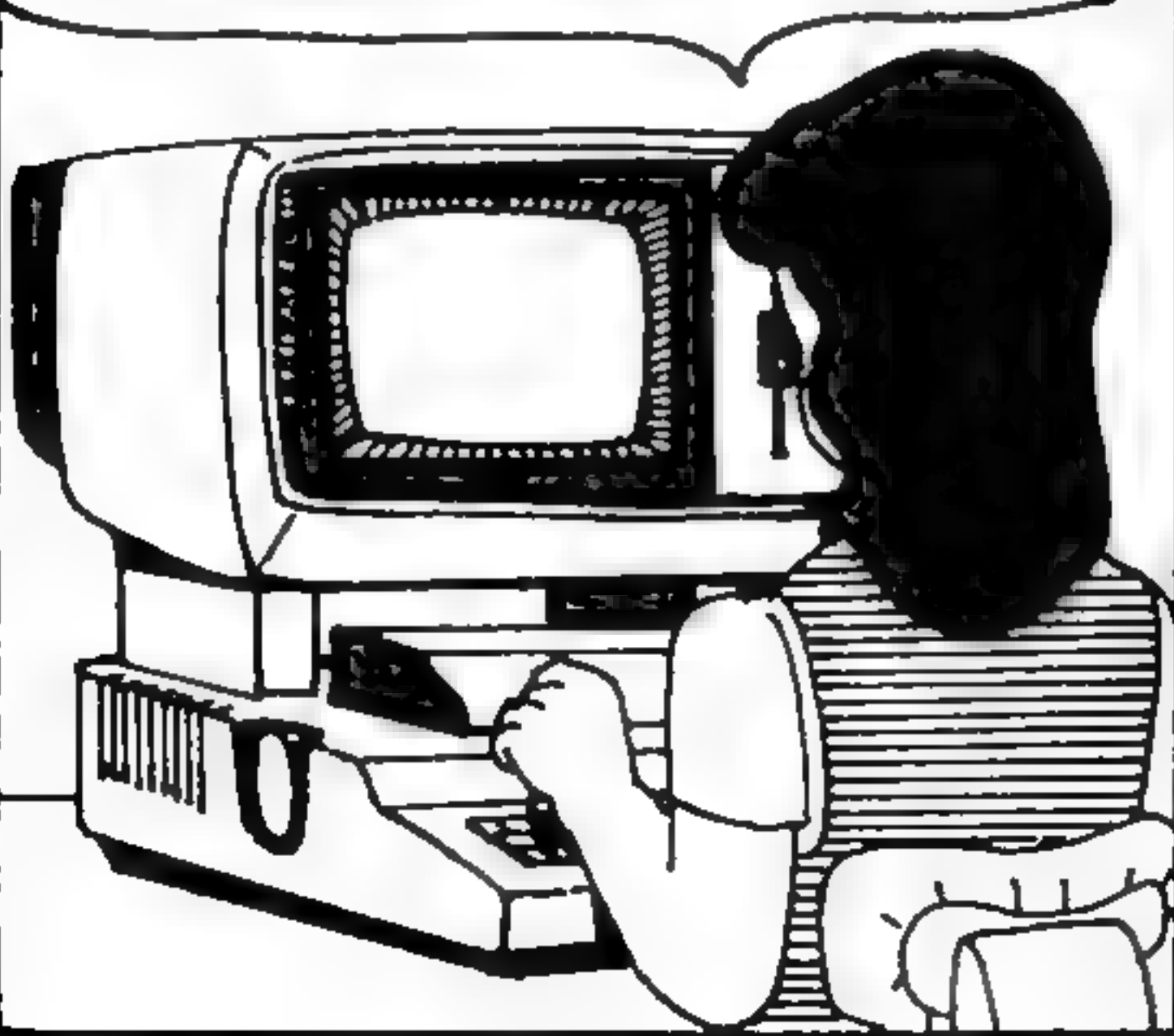
いずれがあやめか  
かきつばた

1

a

A

では、まとめの代わりにサンプルプログラムでこのうちのいくつかの関数を確認してみます。



論理IF文を使うと文字列を思いのままに並べ換えることができます。

IF A\$ > B\$ THEN

ETUKO

TOMOKO

A\$

B\$

1、a、Aの3つのキャラクタはコード化すると、10進数値ではそれぞれ49、97、65となりこれを大きい順に並べると97、65、49です。元のキャラクタに戻すとa、A、1となるでしょう。

(( a ))

A

1



これは画面のハードコピーです。RUNすると=が表示されました。アスキーコード3Dはキャラクタで=になっていますから&H3DはOKです。

```
Ok
AUTO [RETURN]
10 A$=CHR$(&H3D) [RETURN]
20 PRINT A$ [RETURN]
30 END [RETURN]
40 [CAN]
Ok
RUN [RETURN]
=
Ok
```

まず16進コード3Dに&HをつけてA\$=CHR\$(&H3D)がどうなるか、みてみます。

↓ プログラムをキーインする。

```
10 A$=CHR$(&H3D)
20 PRINT A$
30 END
```

このアスキーコードに'&H'をつけてキャラクタコードに換えるテクニックは通信の時かなり応用できるはずなんですが…

VAL

'&H'+'3D'は'&H3D'という文字列になってしまったのです。CHR\$( ) ( )の中は文字列を使うと文法違反です。

さあ困った

これに気を良くして、行番号10をA\$=CHR\$('&H'+'3D')としてRUNしてみました。

```
EDIT 10
10 A$=CHR$("&H"+"3D")
RUN
Type mismatch in 10
OK
```

すると、タイプ・ミスマッチとなってエラーでした。

```
10 INPUT "16進 ASCII コード 2桁 = "; A$
20 B$=CHR$(VAL("&H"+A$))
30 PRINT A$;"...";B$;LPRINT A$;"...";B$;PRINT :LPRINT
40 GOTO 10
```

38...8  
39...9  
3B...;  
3D...  
3F...;

サンプルプリントです。

アスキーコードを16進2桁で入れてそのコードに対応するキャラクタを表示するプログラムです。

ここで浮んだのがVAL関数です。VAL( )の( )の中は文字列だからひょっとしたら…

```
EDIT 10
10 A$=CHR$(VAL("&H"+"3D"))
RUN
=
OK
```

うん、OKです。

アスキーコードと文字列の予備知識はこれぐらいにして、いよいよRS-232Cにチャレンジです。なお次のページには10進数値併記のコード表を載せていますから参考にしてください。

```
Ok
LIST
10 INPUT A$
20 B=VAL("&H"+A$)
30 C$=CHR$(B)
40 C=ASC(A$)
50 PRINT A$,B,C,C
Ok
RUN
RUN
? 45
45
Ok
```

これも画面のハードコピーです。プリントではありません。

アスキーコードをインプットし、まず10進数値を求め、次にキャラクタを求めるものです。行番40では'45'の'4'が10進数値に変換されて52となっています。



# キャラクタ・コード表

ここに載せているキャラクタはASCIIやJISコード指定席です。余白のコードにはメーカーが独自のキャラクタを割り付けていますので、お手持ちのパソコンやニユアルのキャラクタ・コードを見てください。キャラクタAの&H(16進)表現は上桁(4・)と下桁(・1)をつないで&H41となります。&H41=65Dです。

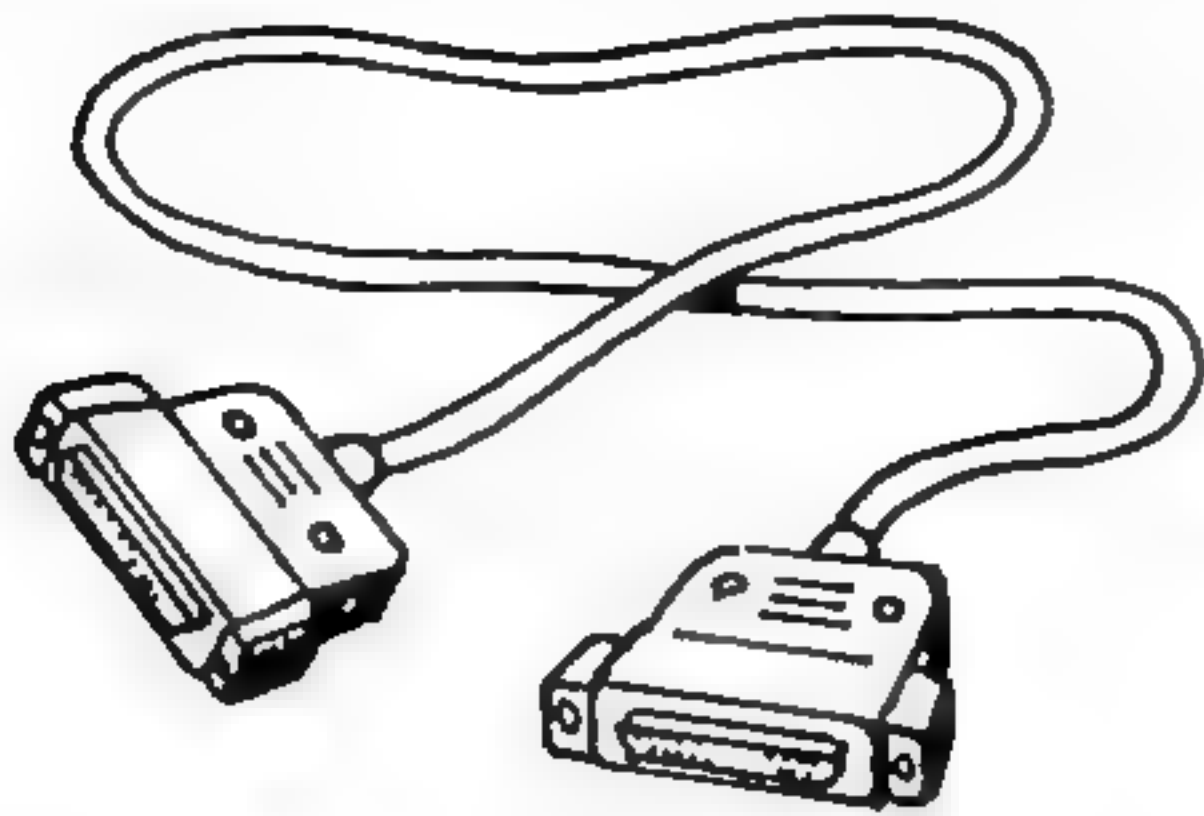
上桁 下桁	0・		1・		2・		3・		4・		5・		6・		7・		8・		9・		A・		B・		C・		D・		E・		F・	
	10進 数値	キャラ クタ	10進 数値	キャラ クタ	10進 数値	キャラ クタ	10進 数値	キャラ クタ	10進 数値	キャラ クタ	10進 数値	キャラ クタ	10進 数値	キャラ クタ	10進 数値	キャラ クタ	10進 数値	キャラ クタ	10進 数値	キャラ クタ	10進 数値	キャラ クタ	10進 数値	キャラ クタ	10進 数値	キャラ クタ	10進 数値	キャラ クタ	10進 数値	キャラ クタ	10進 数値	キャラ クタ
・0	0		16		32	空白	48	0	64	④	80	P	96		112	P	128		144		160		176	-	192	夕	208	ニ	224		240	
・1	1		17		33	!	49	1	65	A	81	Q	97	a	113	q	129		145		161	o	177	ア	193	夕	209	△	225		241	
・2	2		18		34	"	50	2	66	B	82	R	98	b	114	r	130		146		162	ℓ	178	イ	194	ツ	210	×	226		242	
・3	3		19		35	#	51	3	67	C	83	S	99	c	115	s	131		147		163	ℓ	179	ウ	195	テ	211	モ	227		243	
・4	4		20		36	\$	52	4	68	D	84	T	100	d	116	t	132		148		164	`	180	エ	196	ト	212	ヤ	228		244	
・5	5		21		37	%	53	5	69	E	85	U	101	e	117	u	133		149		165	・	181	オ	197	ナ	213	ユ	229		245	
・6	6		22		38	&	54	6	70	F	86	V	102	f	118	v	134		150		166	ヨ	182	カ	198	ニ	214	ヨ	230		246	
・7	7		23		39		55	7	71	G	87	W	103	g	119	w	135		151		167	ヲ	183	キ	199	又	215	ラ	231		247	
・8	8		24		40	(	56	8	72	H	88	X	104	h	120	x	136		152		168	イ	184	ク	200	ネ	216	リ	232		248	
・9	9		25		41	)	57	9	73	I	89	Y	105	i	121	y	137		153		169	ウ	185	ケ	201	ノ	217	ル	233		249	
・A	10		26		42	*	58	:	74	J	90	Z	106	j	122	z	138		154		170	エ	186	コ	202	ハ	218	レ	234		250	
・B	11		27		43	+	59	;	75	K	91		107	k	123		139		155		171	オ	187	サ	203	ヒ	219	□	235		251	
・C	12		28		44	,	60	<	76	L	92		108	l	124		140		156		172	ヤ	188	シ	204	フ	220	フ	236		252	
・D	13		29		45	-	61	=	77	M	93		109	m	125		141		157		173	ユ	189	又	205	ヘ	221	ン	237		253	
・E	14		30		46	・	62	>	78	N	94		110	n	126		142		158		174	ヨ	190	セ	206	ホ	222	"	238		254	
・F	15		31		47	/	63	?	79	O	95		111	o	127		143		159		175	ツ	191	ソ	207	ラ	223	°	239		255	

10進数値と併記しているのでASC関数が使いやすくなります。



# ④まずはハード のつなぎ方

コネクタは25ピン、サイズや信号の電圧レベルなどが規定されていますが、どのピンにどの信号が出ているのかはメーカーやユニットによってまちまちなのが現状です。

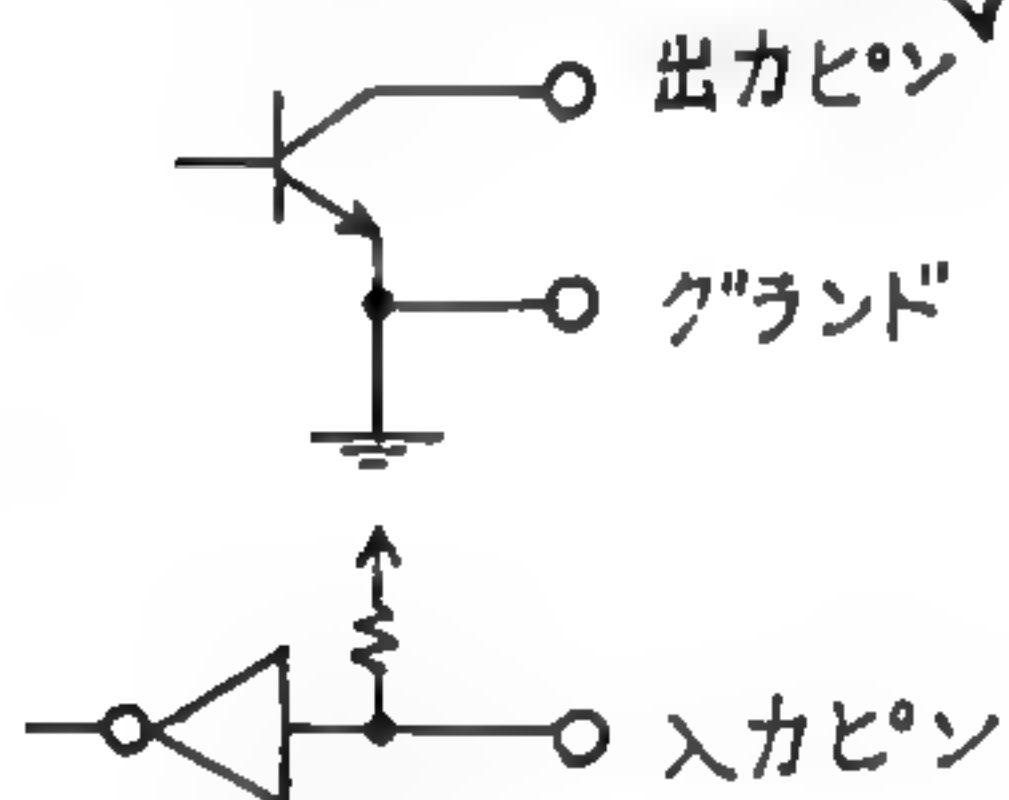


まず、ハード的に何が必要かをみます。コネクタとケーブルが必要なのは当然ですが、ポケコンなどRS-232Cポートがついていないものの場合RS-232Cインターフェイスユニットが必要です。

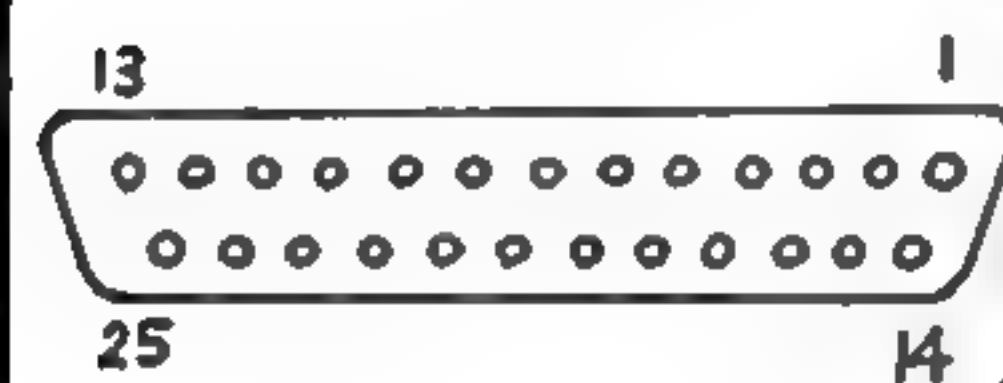
インターフェイス  
CE 158



グラウンド線を除けばあと7つ。これらは出力か入力かのいずれかで、ピンとユニットへの回路の入口は下図のようになっています。



でも使っているピンは25ピンのうちせいぜい8ピンです。信号の持っている意味さえ理解しておけばすぐ対応できます。



そこでコネクタに結線する時は各ユニットの取扱説明書のピン番号と信号の関係をよく見る必要があります。



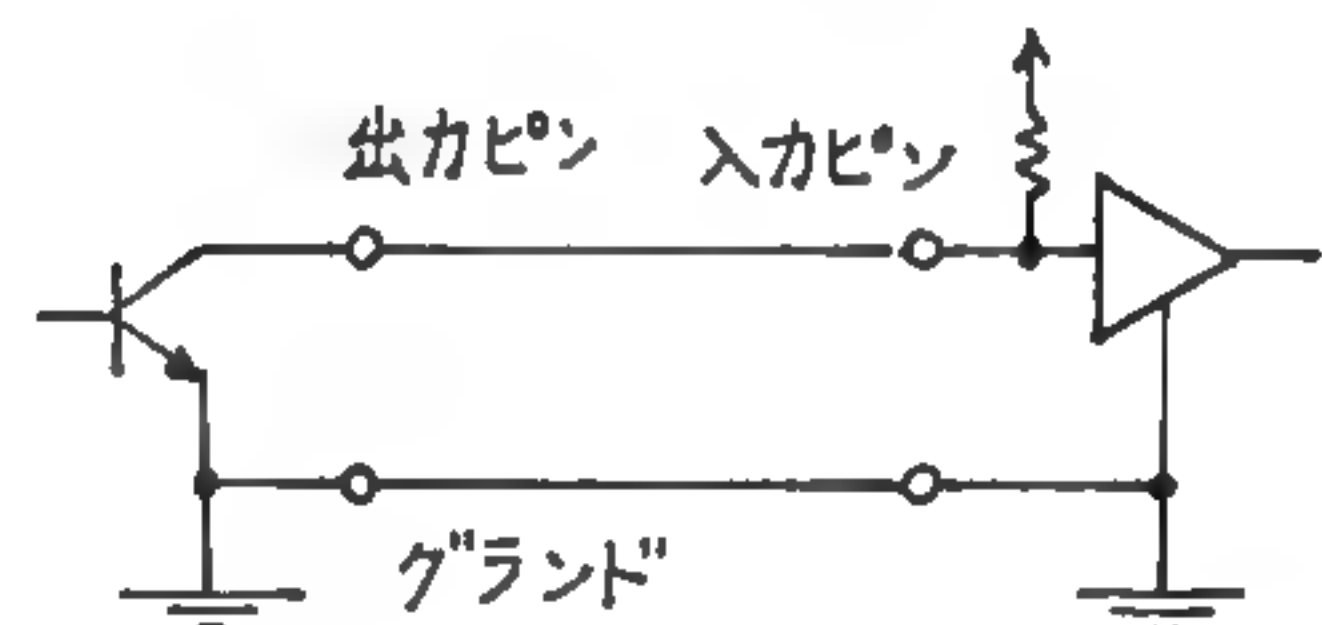
あと2組の受信状態を確認するための信号と、キャリア検出という信号があります。キャリアはモデムを使う時にだけ関係があります。

送信要求...② 送信してくれと要求する。  
送信可...③  
受信要求...④ 受信してくれと要求する。  
受信可...⑤  
キャリア検出...

データ用のピンは送信用、受信用それぞれ1個ずつあります。こちらの送信側は相手の受信側に接続します。



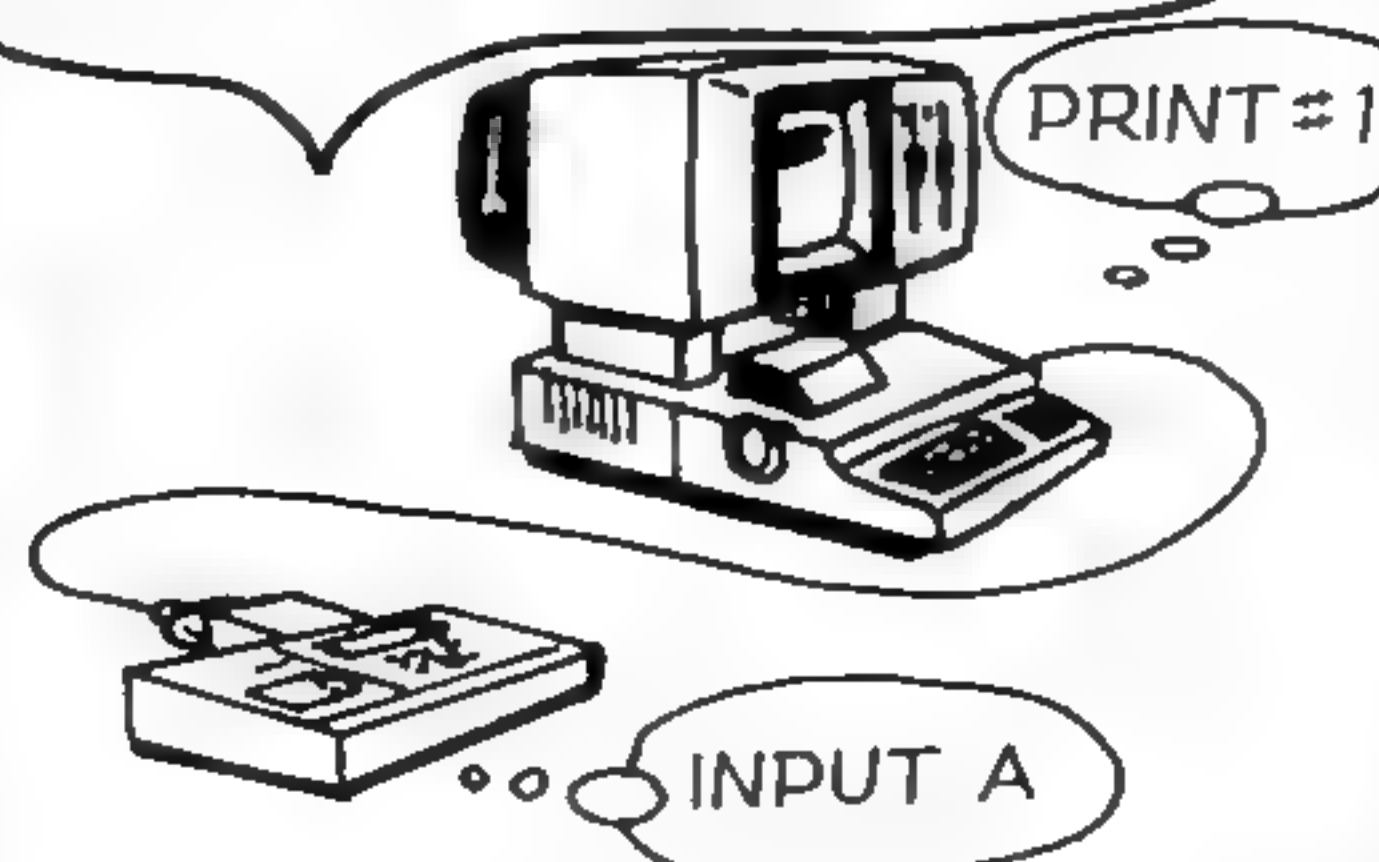
こちらの出力ピンは相手の入力ピンに、相手の出力ピンはこちらの入力ピンに接続します。



でもこんなことは結線時にちょっと注意しておけばよいだけです。BASICレベルのプログラムではまったく意識する必要がありません。



プログラムでは双方同時に送信要求や受信要求が出るとは考えられません。片方がデータを送るタイミングではもう一方はデータを受信するようにプログラムされているはずで



送信要求はINPUTのような受信文が実行された時、受信要求はPRINTのような送信文が実行された時それぞれ出ていくと考えられます。



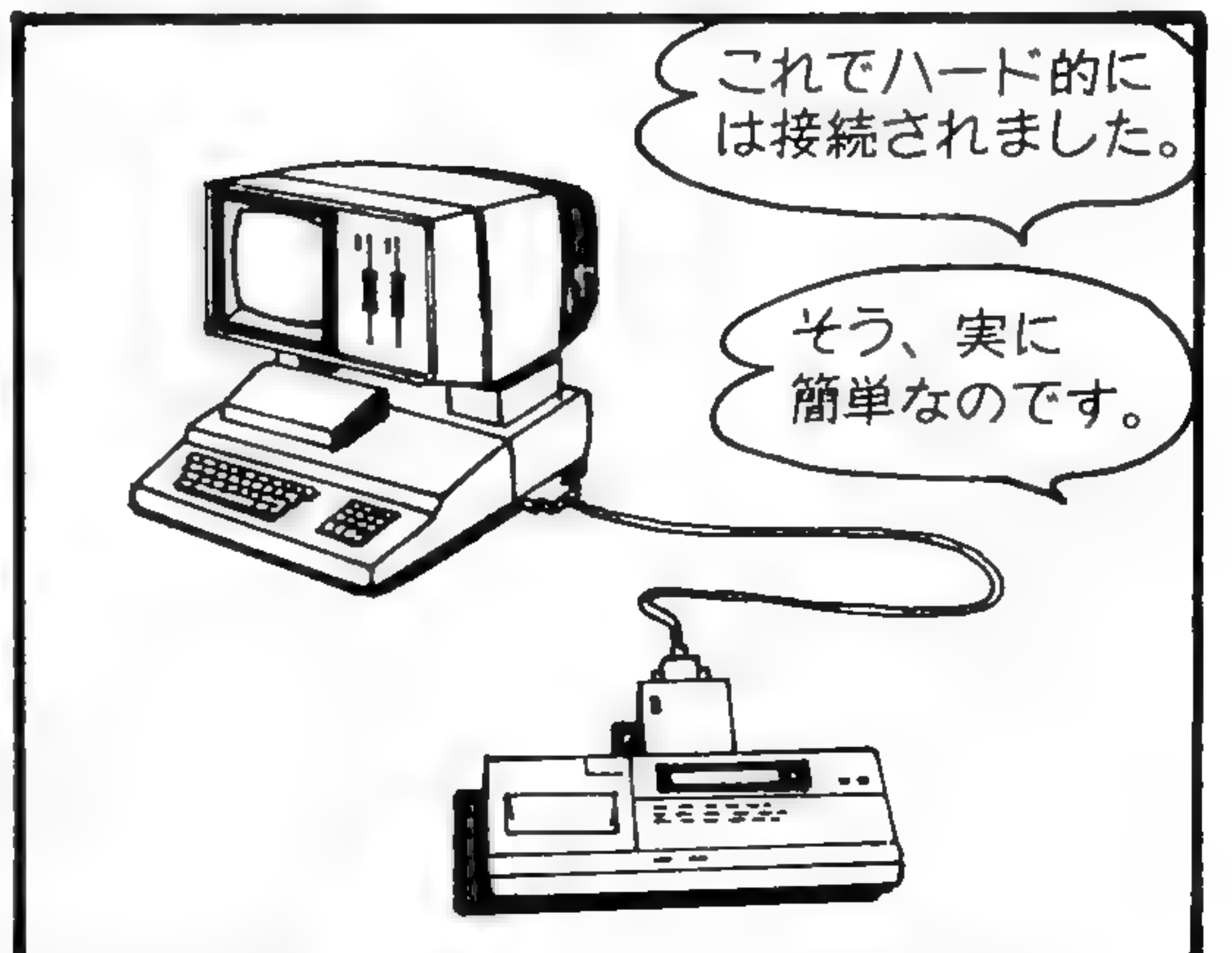
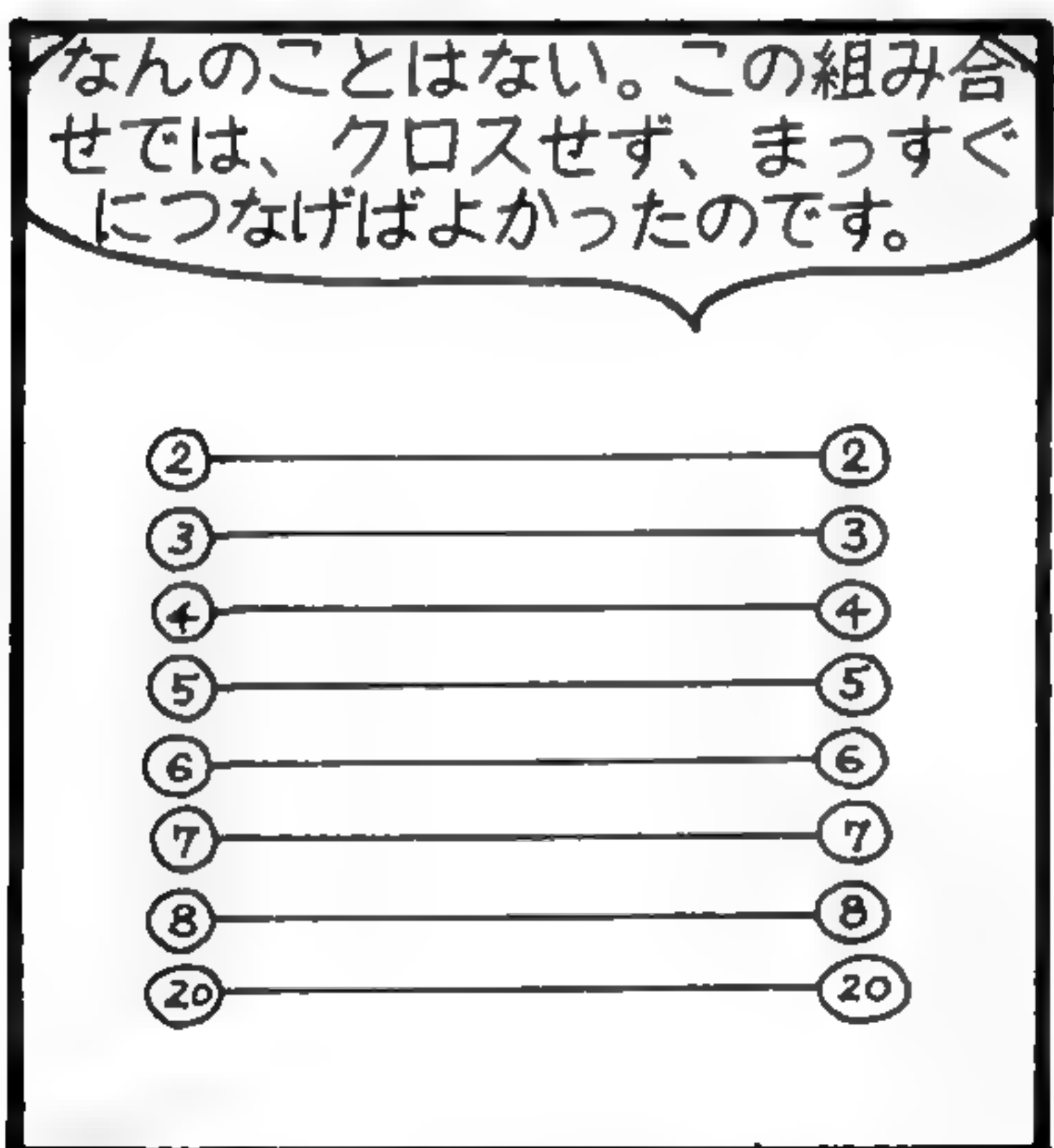


PC1500側			
送信データ	--- TD	出力	②
受信データ	--- RD	入力	③
送信要求	--- RTS	出力	④
送信可	--- CTS	入力	⑤
データ セットレディ	--- DSR	入力	⑥
信号用接地	SG		⑦
キャリア検出	--- CD	入力	⑧
端末装置 レディ	--- DTR	出力	②⑩

2つのマニュアルの中の  
信号名をそのまま転記し  
ました。



IF800-30側			
② 送信データ	--- BA	入力	
③ 受信データ	--- BB	出力	
④ 送信要求	--- CA	入力	
⑤ 送信可	--- CB	出力	
⑥ 受信要求	--- CC	出力	
⑦ 信号グランド			
⑧ キャリア検出	--- CF	出力	
②⑩ 受信可	--- CD	入力	

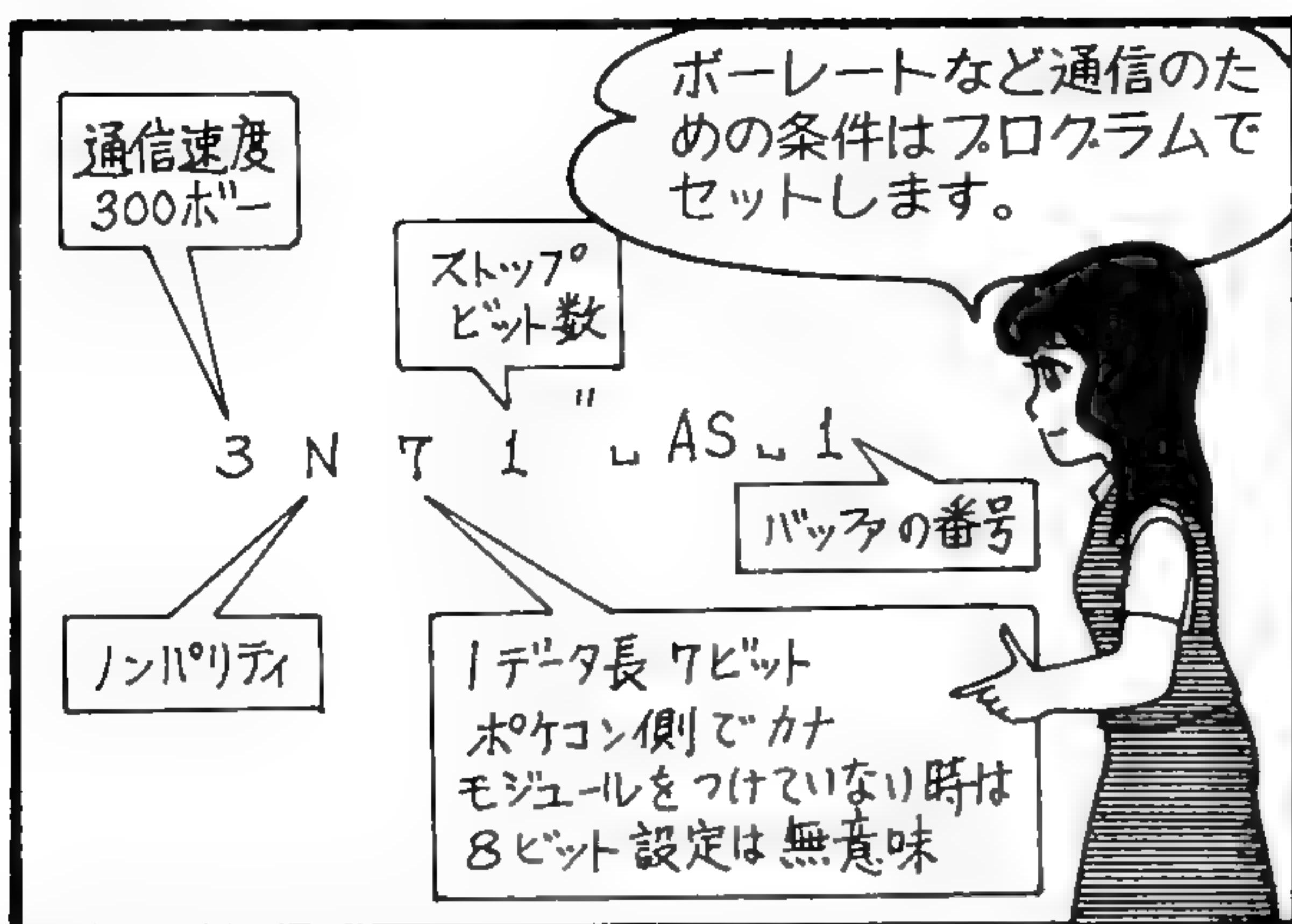




## ⑤ RS-232C 活用のソフト

AUTO  
10 OPEN "COM1:3N71" AS 1

まず通信条件の宣言です。ここはパソコンによってそれぞれ書き方が違います。



文50と60でF\$の内容を見てプログラムを続行するか終了するかの判断をしています。F\$の内容がENDだったら終了です。

```
50 IF F$ <> "END" THEN 30
60 PRINT "TEST END"
70 END
```

文40は変数F\$の内容を表示する命令です。これでポケコンから送られたデータの内容を見ることができます。

```
40 PRINT F$
```

文30はバッファレジスタ#1から変数F\$にデータを取り出す命令です。

```
20 PRINT "RS-232C TEST PROGRAM"
30 INPUT #1, F$
```

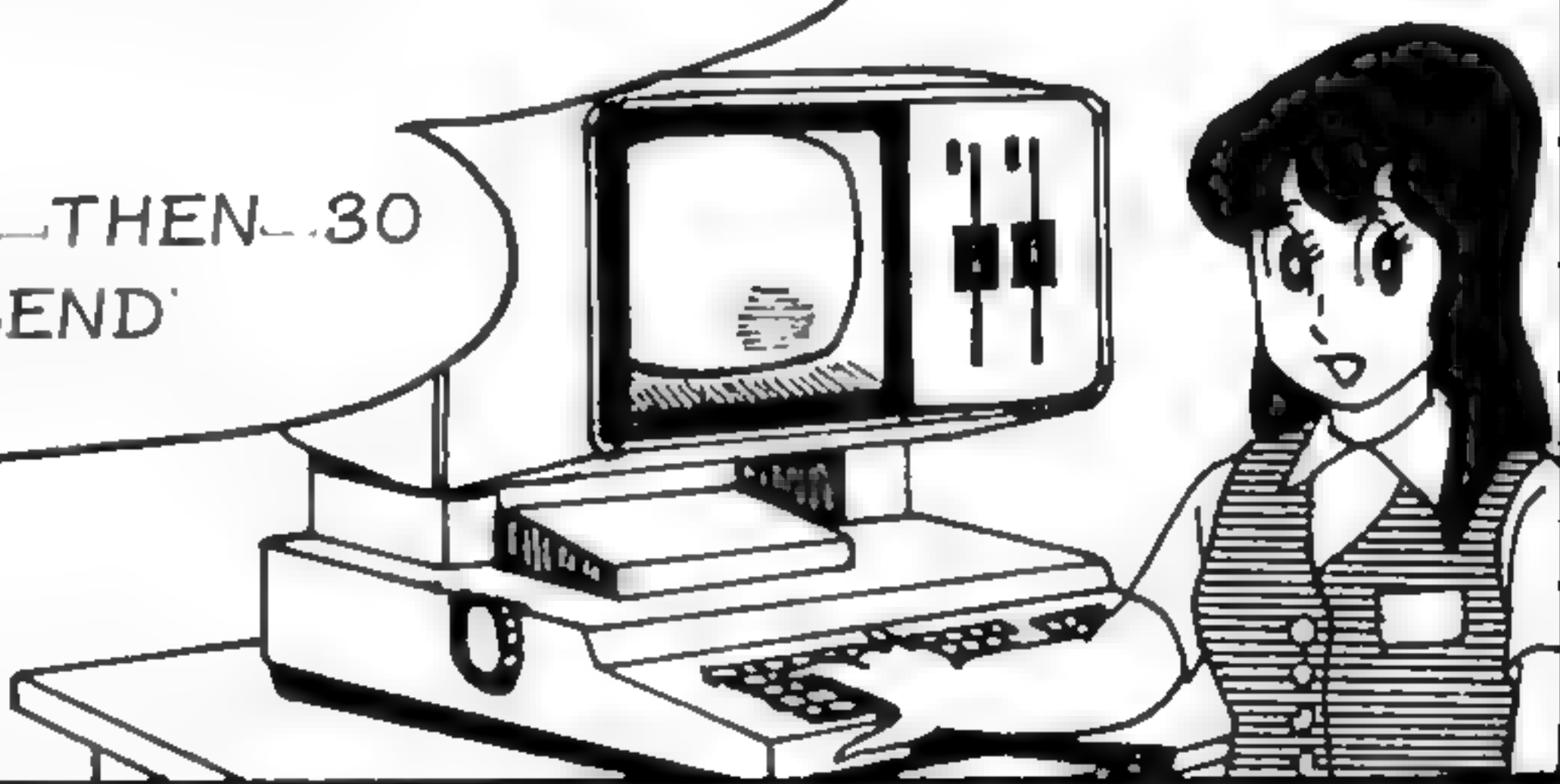
次はポケコン側です。命令文はポケコンのマニュアルを参照しながら、プログラムします。



```
10 OPEN "COM1:3N71" AS 1
20 PRINT "RS-232C TEST PROGRAM"
30 INPUT #1, F$
40 PRINT F$
50 IF F$ <> "END" THEN 30
60 PRINT "TEST END"
70 END
```

RUN RETURN

RUNすると、IF側はデータ待ちとなります。



文30は2つの出力RTSとDTRを共にONにする命令です。アウトポートのステータスを決める文です。アウトスタットとも読んでください。

```
30: OUTSTAT = 0
```

次はセットデバイス文でPOをセットします。これに対応する命令はLPRINT文です。セットデブと読みましょう。

```
20: SETDEV = PO
```

解除する時は、SETDEVだけの文にすればよいのです。解除後は印字文となります。

まず、通信条件の設定はセットコミュニケーション文で行ないます。セットコムとも呼びましょうか。

```
10: SETCOM = 300, 7, N, 1
```

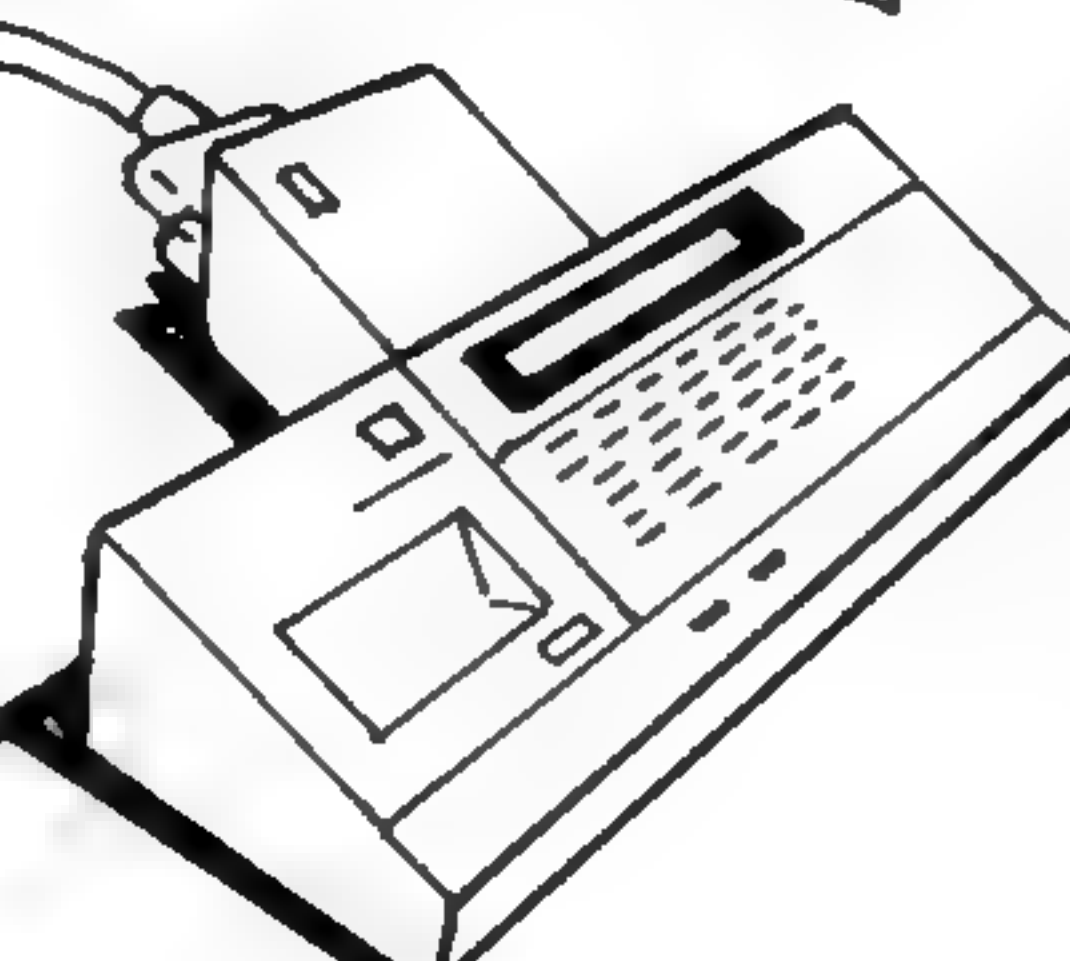
IF側と同条件になるようにセットしなければなりません。



以下PC-1500からRS-232Cポートに出力する命令を書きます。

```

5: WAIT 200
10: SETCOM 300, 7, N, 1
20: SETDEV PO
30: OUTSTAT 0
40: INPUT DATA=: A$
50: LPRINT A$
60: IF A$ <> "END" THEN 40
70: PRINT "TEST END"
80: END
    
```

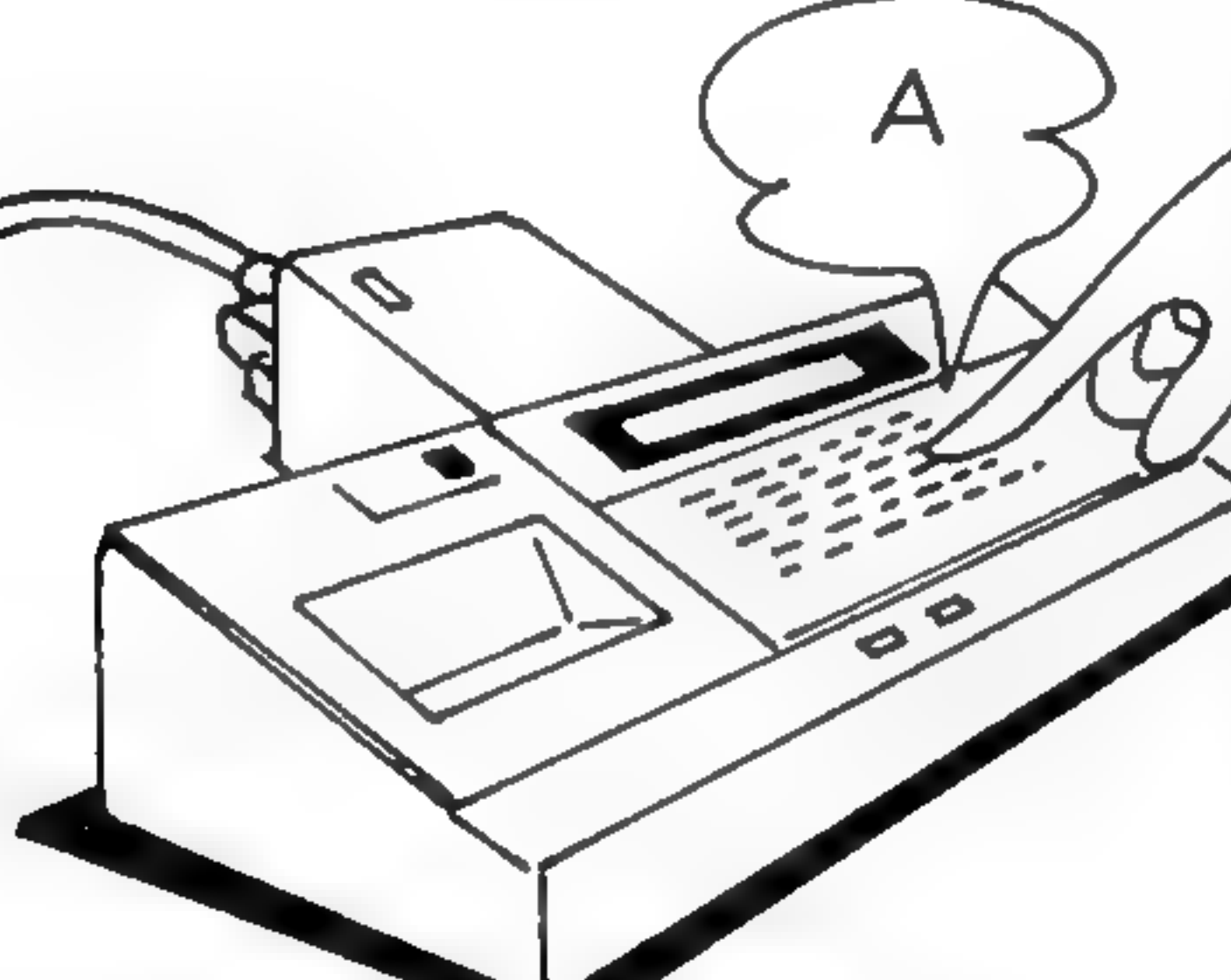


ポケコンの場合SETDEV文が実行されると、その後のINPUTやLPRINTなどの命令はRS-232Cポートに対して行なわれます。

アサイン	方向	命令文
KI	(INPUT)	INPUT
PO	(OUTPUT)	LPRINT

まずA **ENTER** です。

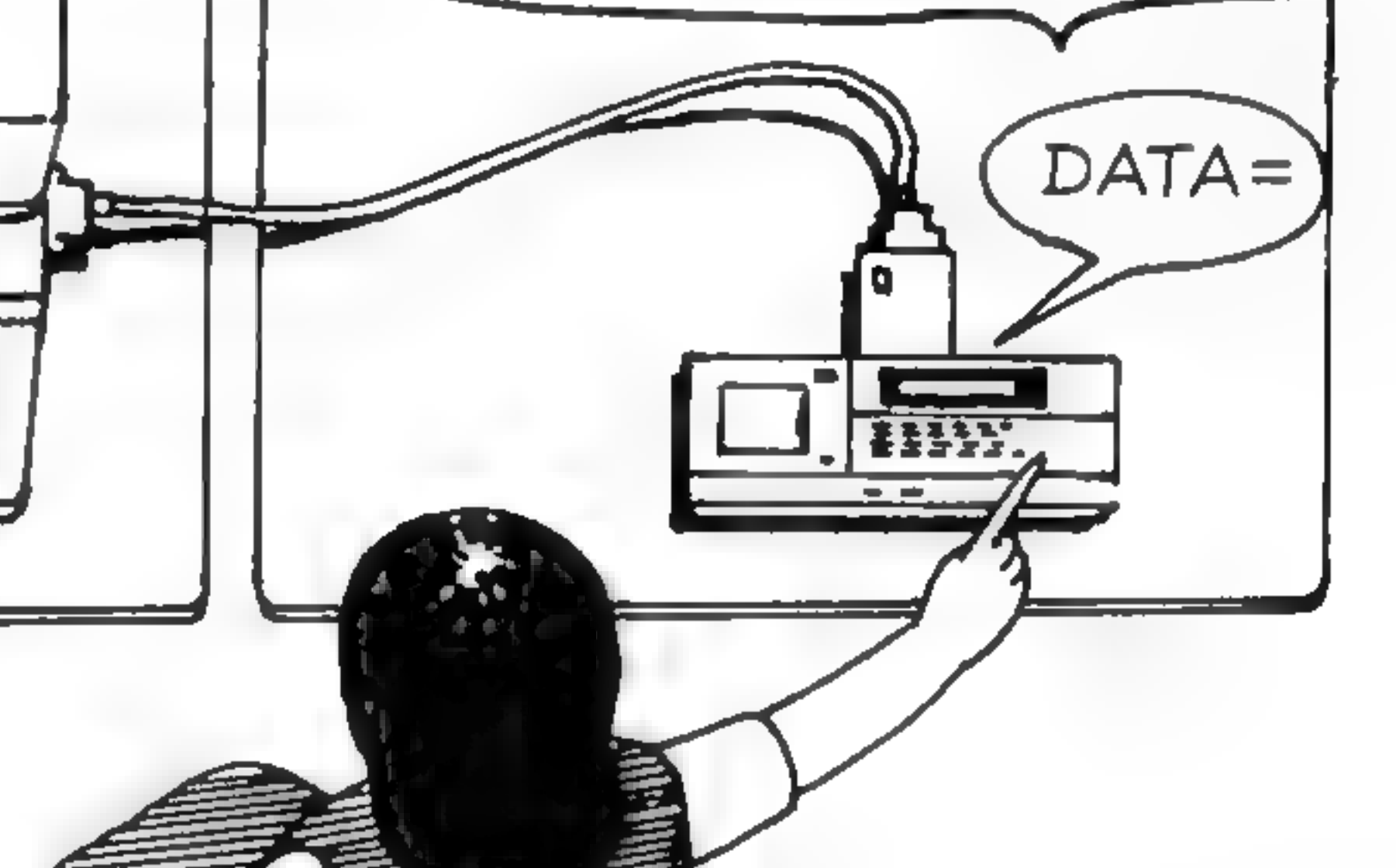
A



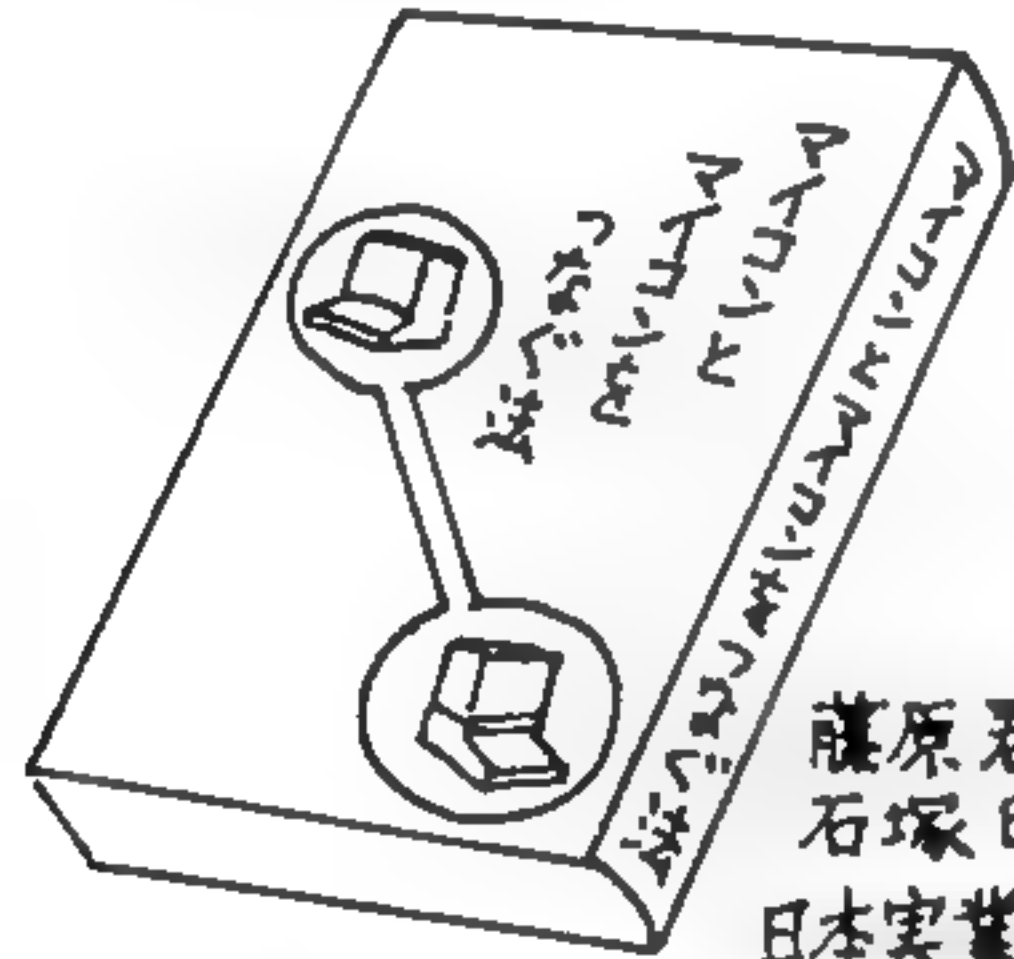
RS-232C TEST PROGRAM

これでポケコンPC-1500とIF800-30はハード、ソフト共に接続完了しました。

DATA=



もっと、プログラムの説明をしてもいいのですが、いい本がすでに出ていますので省略しましょう。



藤原君鬼  
石塚日出子 著  
日本実業出版社

次に **LOVE ETUKO** **ENTER** と入れてみました。

```

RS-232C TEST PROGRAM
A : : : : :
: I LOVE ETUKO :
: : : : :
    
```

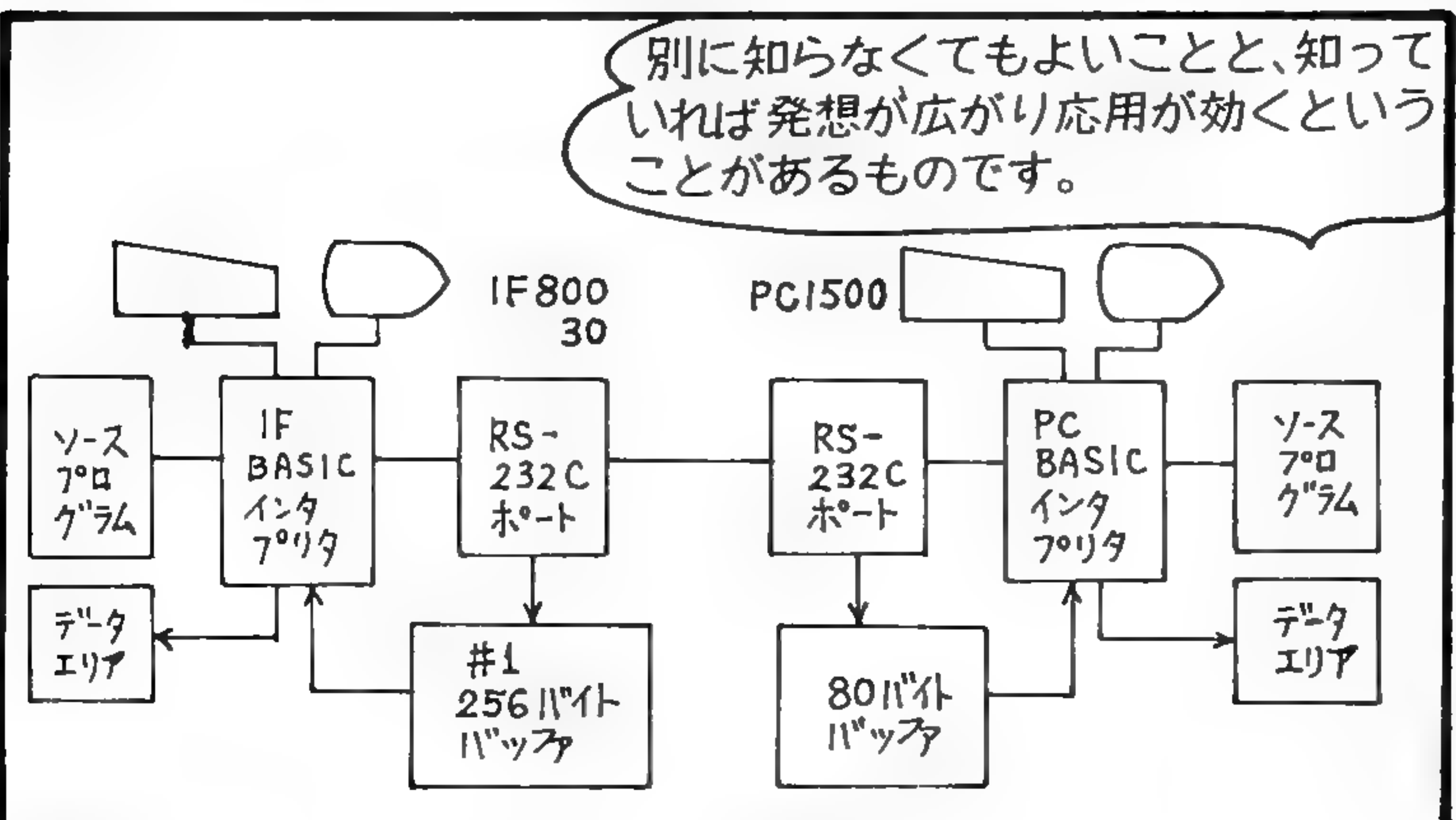
やりました。  
これもOKです。

そして、IFの画面を見ると...

```

RS-232C TEST PROG
A
    
```

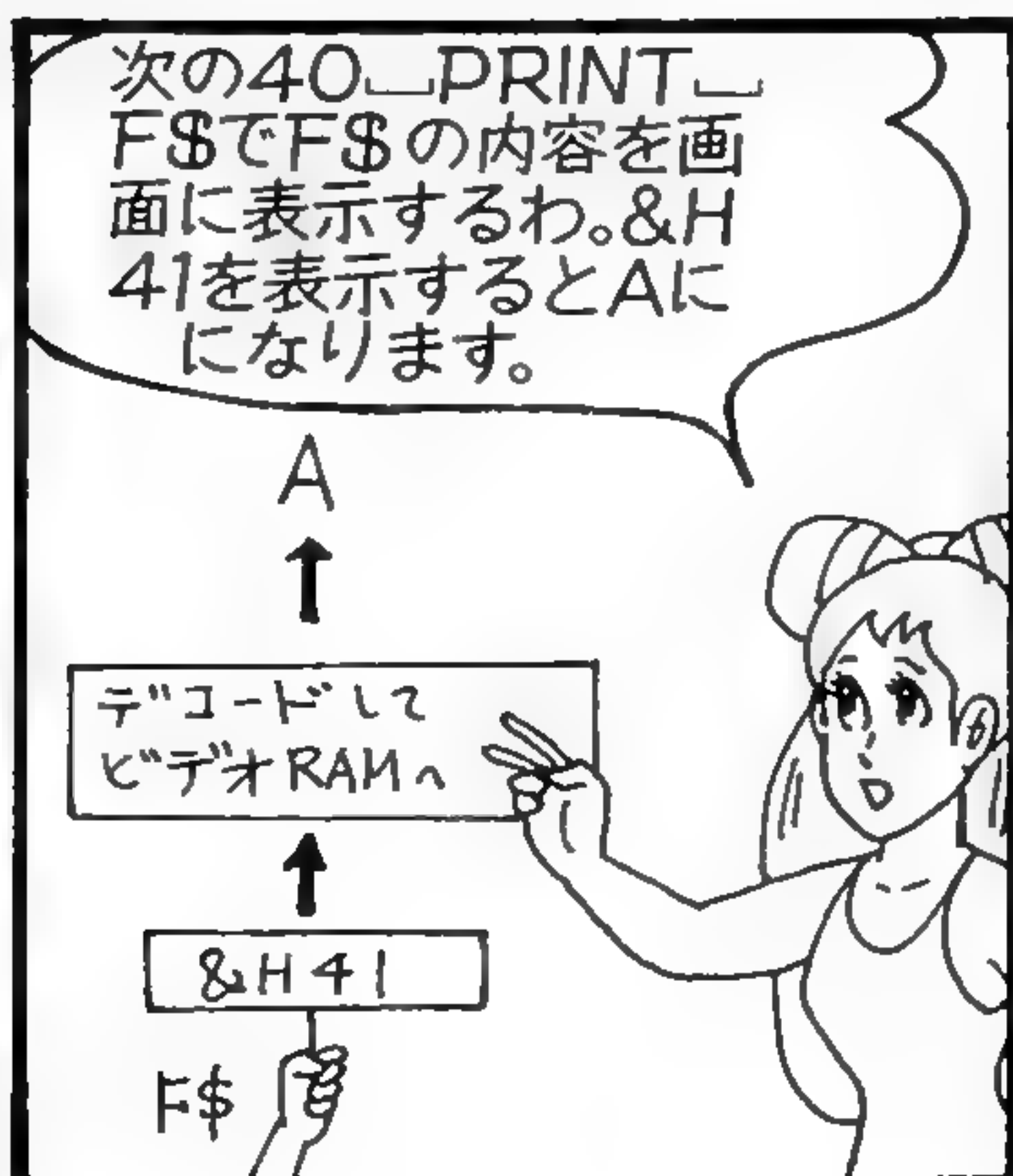
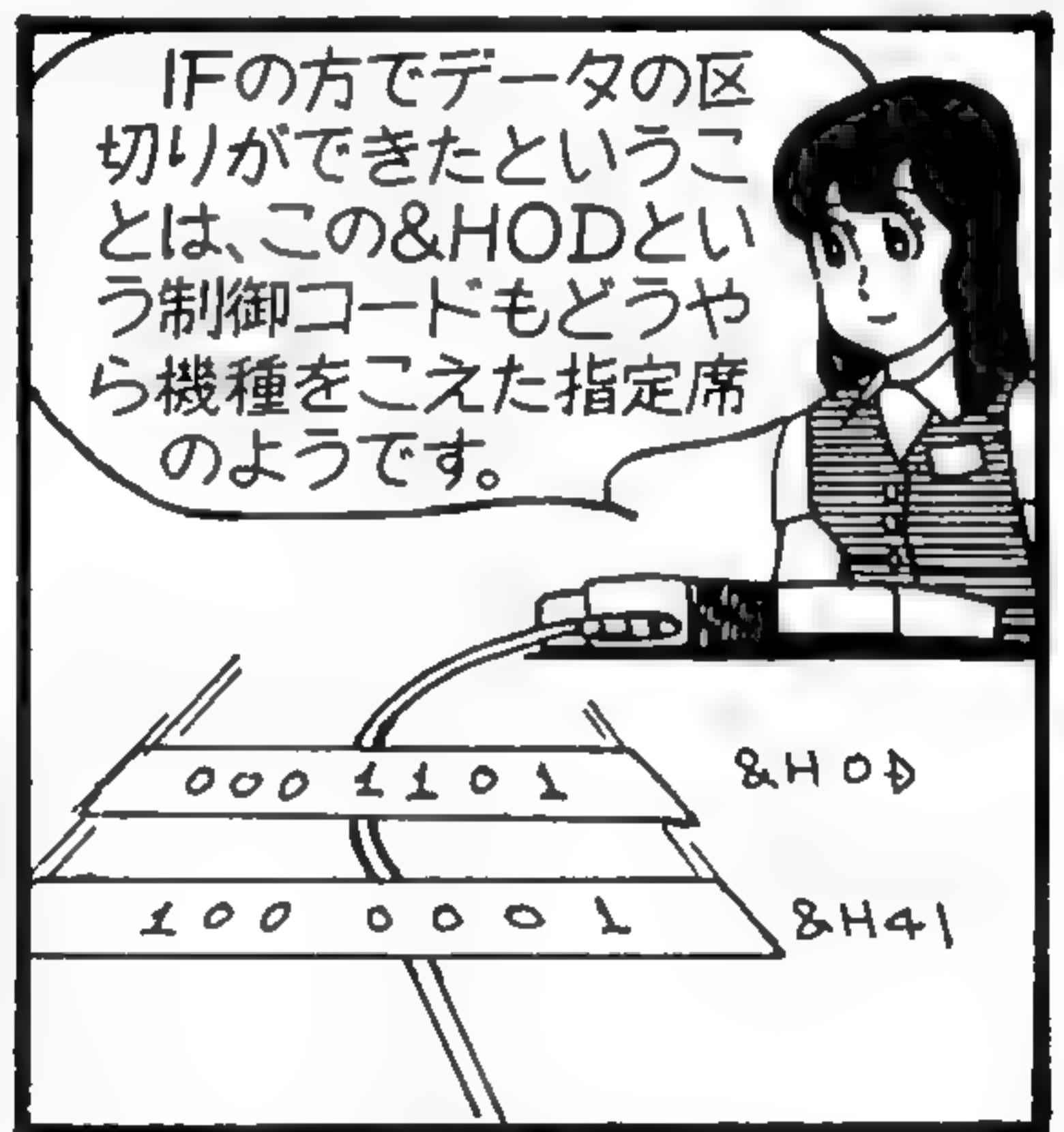
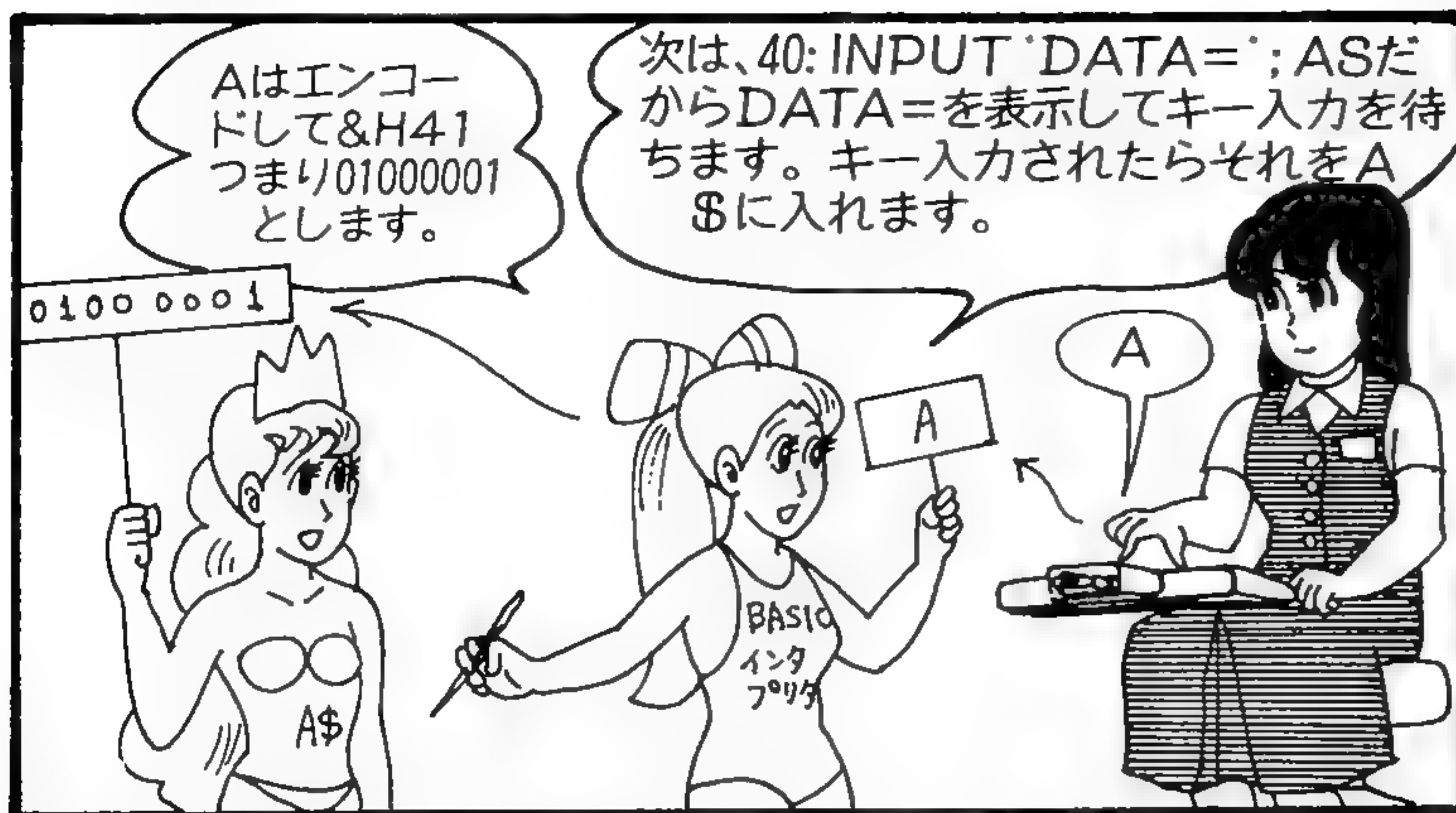
やった。PC側で  
入れたAが表示されま  
した。



そこで、ここではソフトメカニズムについてもう少し考えてみたいのです。









## ⑥データの転送

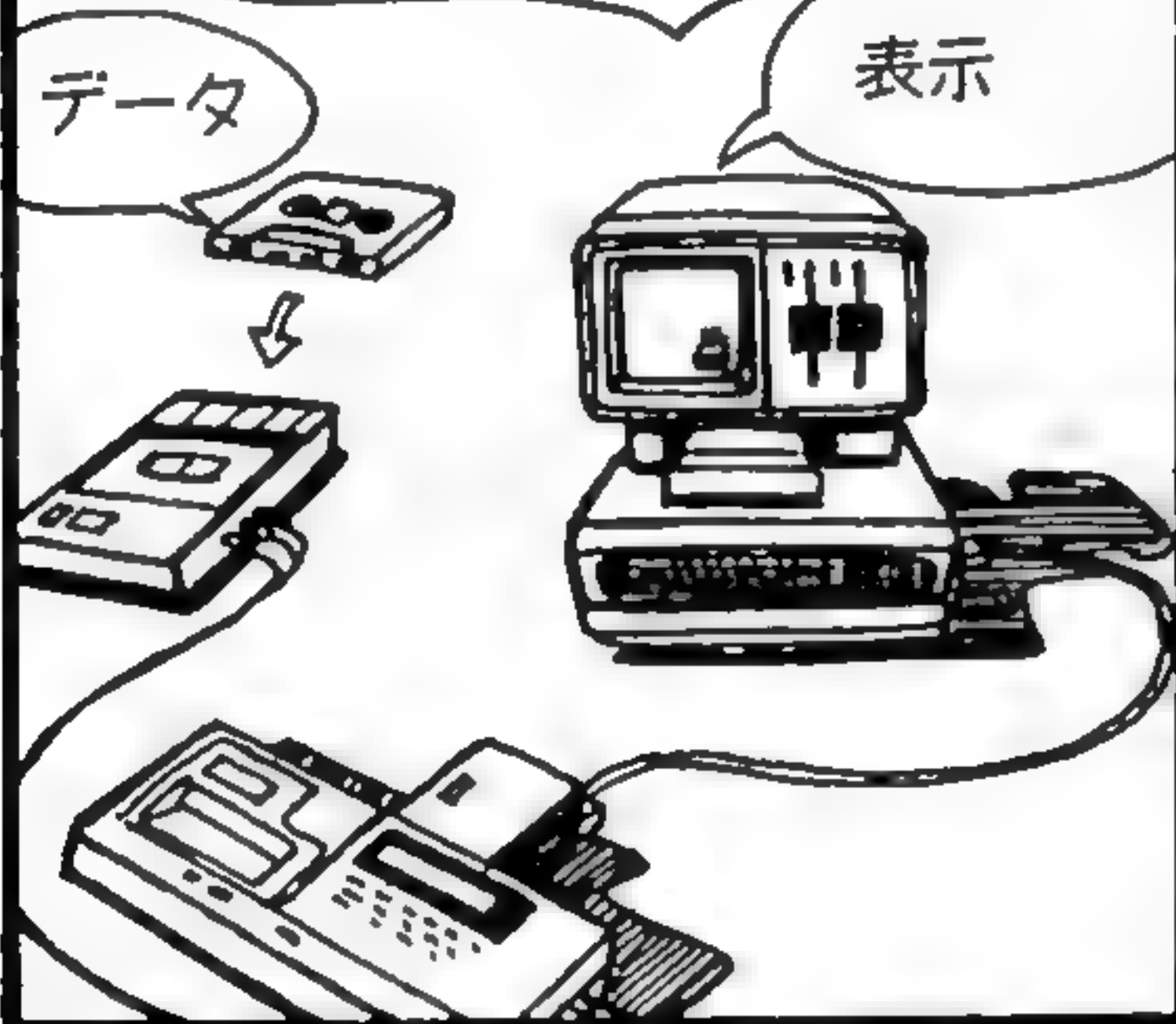
ポケコンやハンドヘルドの場合は価格的な手軽さからカセットにデータのファイルを作ることが多いと思います。この場合はデータが順番に並ぶシーケンシャルファイルになります。



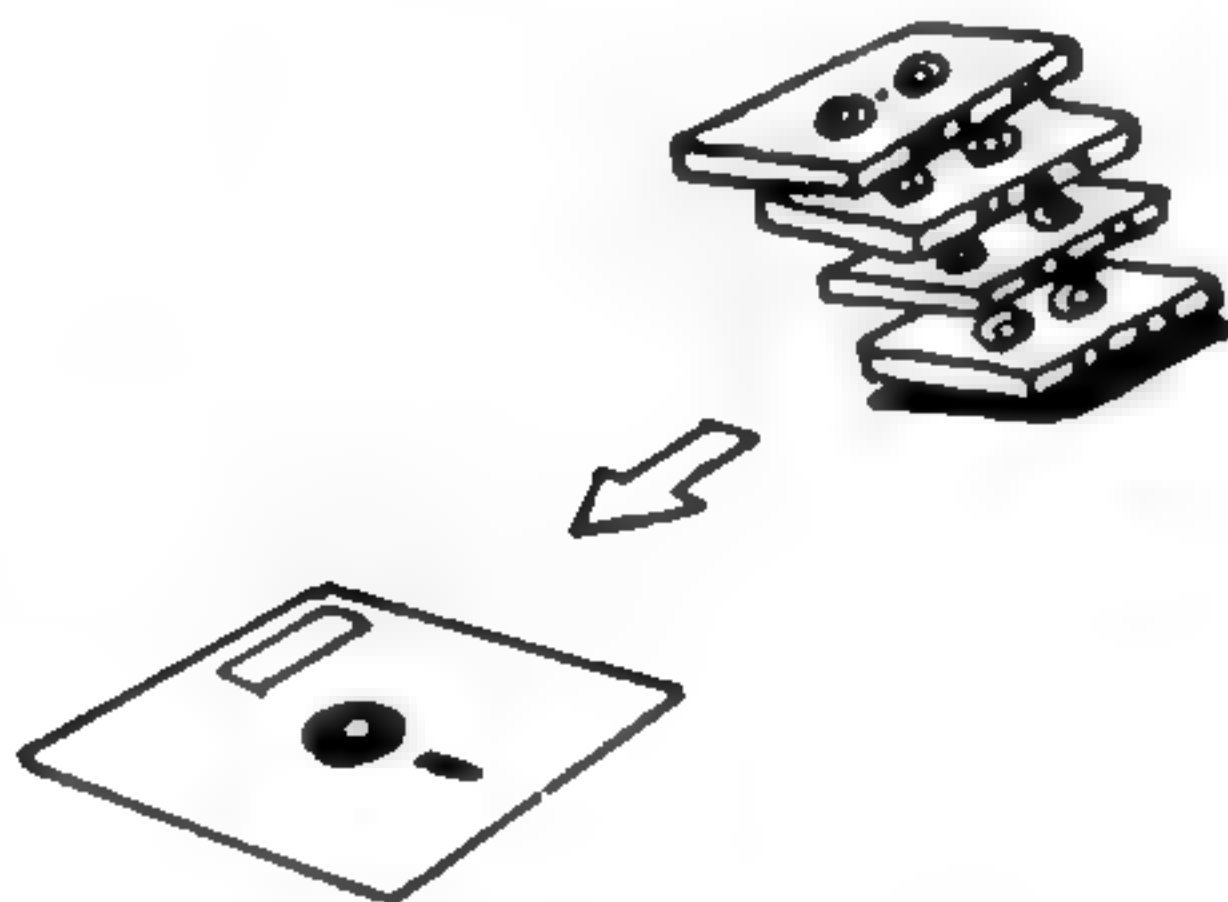
ころんで送るということじゃないのよ。データを送る時は転送という言い方をするんです。



ここではカセットの中のデータをポケコンに呼び出して変数の中に入れそれをRS-232Cでパソコンに送る手続きをやってみます。



そこで最終的にはカセットに作ったファイルをフロッピー(FD)に移しかえる手続きが必要になると思います。



ただ、カセットのファイルではデータの集計や編集に時間がかかったり何回もカセットを交換しなければならなかったりして大変です。



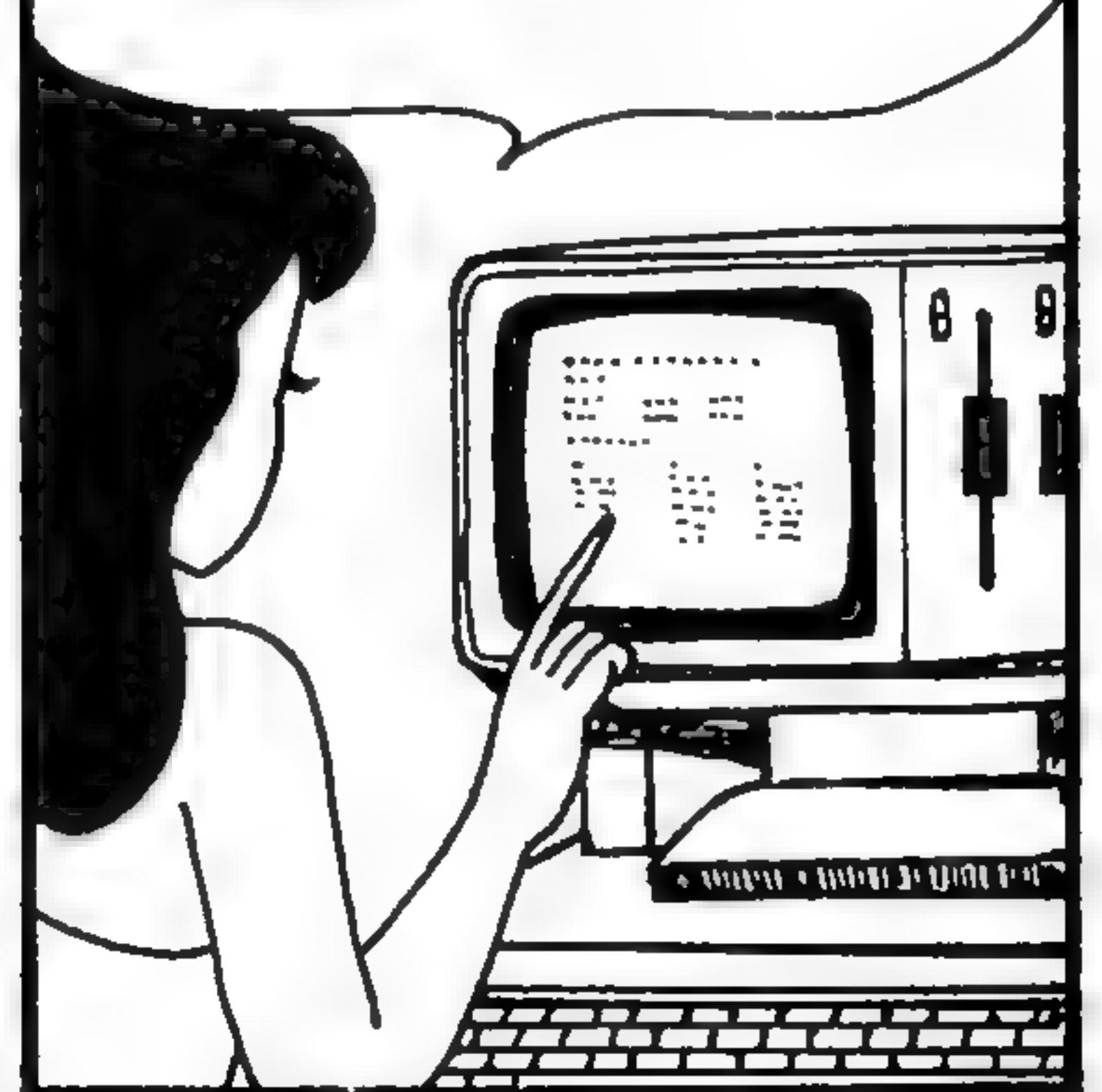
C  
D  
E  
N  
:  
:  
:  
G  
C\$  
A2(N,2)

ポケコンで  
使用した  
変数名

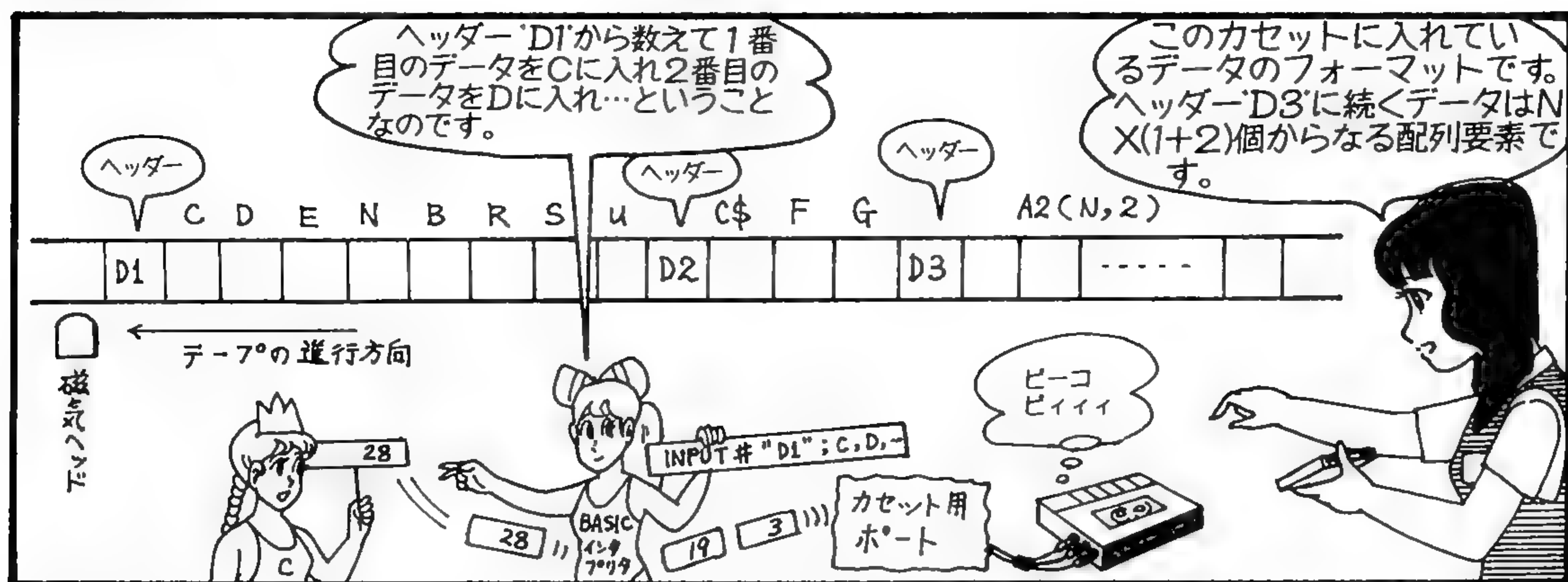
カセットの中には数値と文字と配列がひとつのレコードとして入っています。



パソコン側では送られたデータをCRTに表示します。ランダムファイルの作り方は、A章でお話していますのでここでは省略します。







こちらはIF800-30側のプログラムリストです。必要なデータの先頭がどれなのか決めるためにヘッダー 'ABC' を使いました。ヒントはシャープのインターフェースCE-158のマニュアルの記事から得ました。

```

10 'S59/01/17 FILE NAME RS232C3
20 OPEN "COM1:3N71" AS 1
30 PRINT #1, "ABC"
40 CLS:PRINT "PC-1500 TO IF800-30 BY RS-232C"
50 INPUT #1, Y$:PRINT Y$
60 IF RIGHT$(Y$, 3) = "END" THEN 260
70 IF RIGHT$(Y$, 3) <> "ABC" THEN 50
80 PRINT #1, "ABC"
90 INPUT #1, C, D, E, N, B, R, S, U, F, G
100 PRINT USING "C=####, D=####, E=####"; C, D, E
110 PRINT USING "N=####, F=####, G=####"; N, F, G
120 INPUT #1, C$
130 PRINT "C$="; C$
140 IF Z=1 THEN 160
150 DIM A2(N, 2):Z=1
160 FOR J=0 TO 2
170 FOR I=1 TO N
180 INPUT #1, A2(I, J)
190 NEXT I
200 NEXT J
210 PRINT "I", "J=0", "J=1", "J=2"
220 FOR I=1 TO N
230 PRINT I, A2(I, 0), A2(I, 1), A2(I, 2)
240 NEXT I
250 X=1:FOR I=1 TO 15000:X=X*X:NEXT I:GOTO 40
260 PRINT "PC-1500 TAPE END":END

```

これはポケコン側のプログラム・リストです。カセットからデータを読み込み、それをIF-800-30に転送するものです。

```

10:REM PC-1500 TO
    IF800-30 BY
    RS-232C
20:REM S59/01/09
95:WAIT 50:Z=0
100:INPUT # "D1"; C,
    D, E, N, B, R, S, U
120:INPUT # "D2"; C$,
    F, G
140:IF Z=1 THEN 200
160:DIM A2(N, 2):Z=
    1
200:INPUT # "D3"; A2
    (*)(214ページ参照)
300:SETCOM 300, 7, N
    , 1
305:OUTSTAT 0
310:"A":SETDEV PO:
    IF N<>1 THEN 32
    0
315:LPRINT "END":
    WAIT:PRINT "T
    APE END":END
320:LPRINT "ABC"
325:WAIT 10
330:SETDEV KI:
    INPUT Y$:PRINT
    Y$:IF RIGHT$(
    Y$, 3) <> "ABC"
    THEN 330
350:SETDEV PO
400:LPRINT C, D, E, N
    , B, R, S, U, F, G
420:LPRINT C$
430:FOR J=0 TO 2
440:FOR I=1 TO N
450:P=A2(I, J)
460:LPRINT P
470:NEXT I
480:NEXT J
490:CLS:GOTO 100

```

'D3' に続くデータの配列です。要素の大きさに変数Nを使っているのが 'D3' を読み込む前に DIM(N, 2) を宣言する必要があります。文140は2回以降再宣言するのをさけるためのものです。

```

140: IF Z=1 THEN 200
160: DIM A2(N, 2):Z=1
200: INPUT # "D3"; A2(*)

```

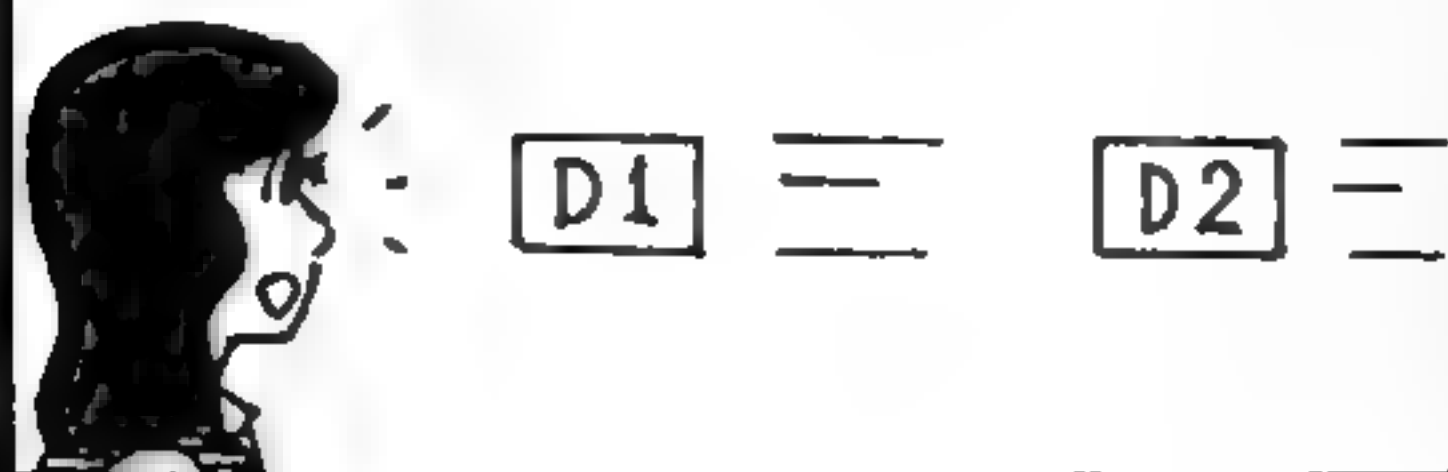


まずポケコン側からこのリストの意味を説明しましょう。テープからの読み込み命令は INPUT #、このリストではヘッダー 'D1'、'D2'、'D3' をつけてデータの先頭を検出するようにしています。

```

100: INPUT # "D1"; C, D, E,
    N, B, R, S, U
120: INPUT # "D2"; C$, F, G

```





SETDEV PO(セットデブポートアウト)は、LPRINT(エルプリント)文で出力するためのものです。

SETDEV PO

⋮  
LPRINT

RS-232Cへの出力文となる

⋮  
SETDEV

RS-232Cへの入出力の解除

⋮  
LPRINT

プリンタへの出力文となる

'A'はラベルです。DEF Aのキー操作でプログラムがここから走ります。この、ラベルを使つてのプログラムスタートでは数値がクリアされないでテストの時便利です。

310: "A": SETDEV PO

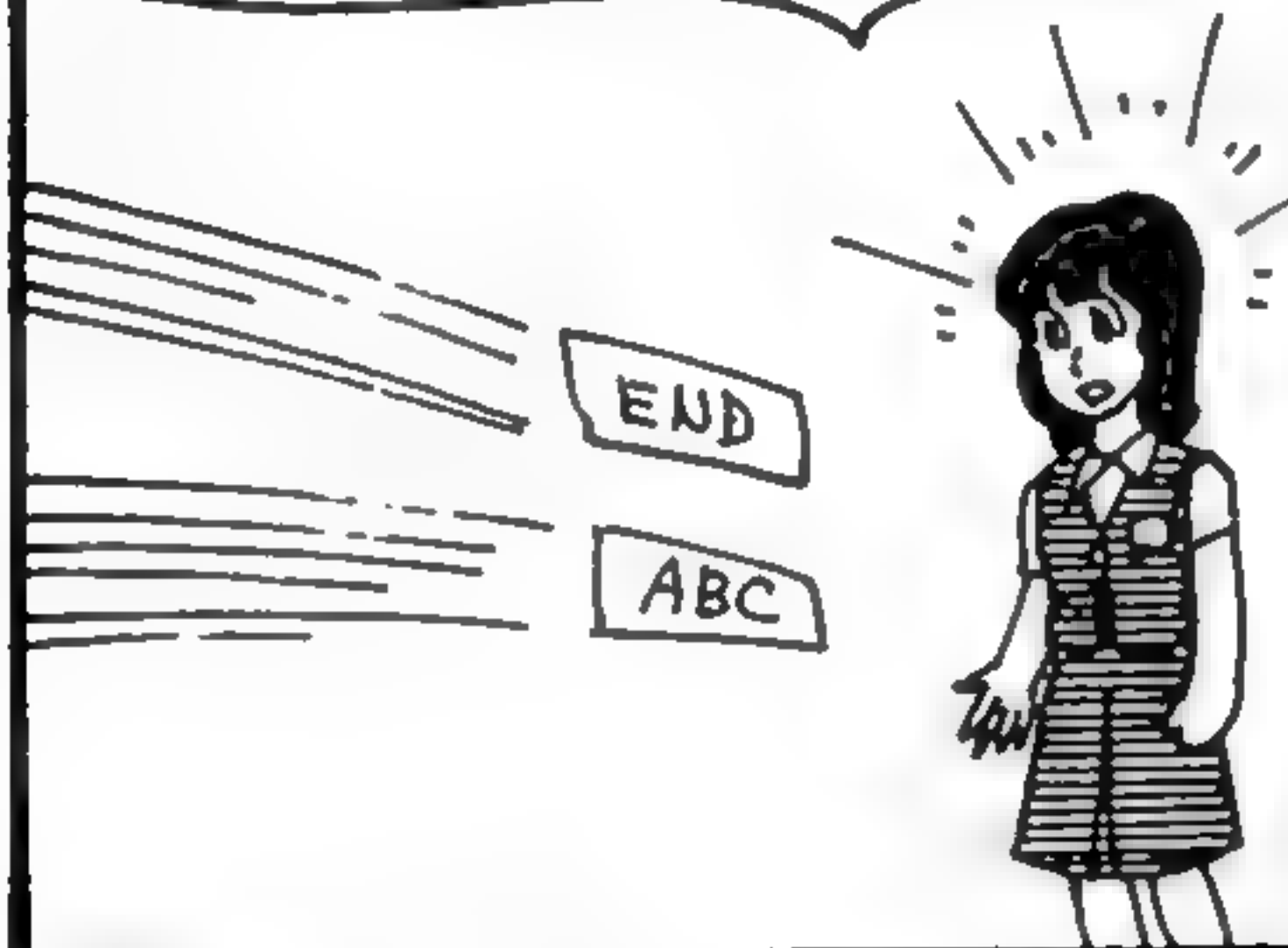
データ1レコード分をポケコンの変数に入れ終わったら、次は転送の用意です。まず、コミュニケーション条件のセットです。

300: SETCOM 300,7,N,1  
305: OUTSTAT 0

プログラム終了ルーチンです。RS-232Cポートに'END'を出し、TAPE ENDと表示します。ENTERを押すと、>(フロンフト表示)になってプログラム終了です。

315: LPRINT "END": WAIT:  
PRINT "TAPE END":  
END

'END'は、IF側にプログラム終了を知らせるためのキーワード。ABCはデータ転送に先がけて送るヘッダーです。



カセットの最終レコードにはNに1を入れてあります。N<>1はNが1でなかったら、という意味です。この<>(でなかったら...)は便利です。

IF N<>1 THEN 320

もしもNが1でなかったら320へ、1なら次の文へ...



ポケコンからの'END'または'ABC'を受けとるIF800-30側のプログラムです。

50 INPUT #1, Y\$: PRINT Y\$  
60 IF RIGHT\$(Y\$, 3) = "END" THEN 260  
70 IF RIGHT\$(Y\$, 3) <> "ABC" THEN 50  
80 PRINT #1, "ABC"

その対策としてはやっぱりヘッダーを送るしかないなと思ったの。



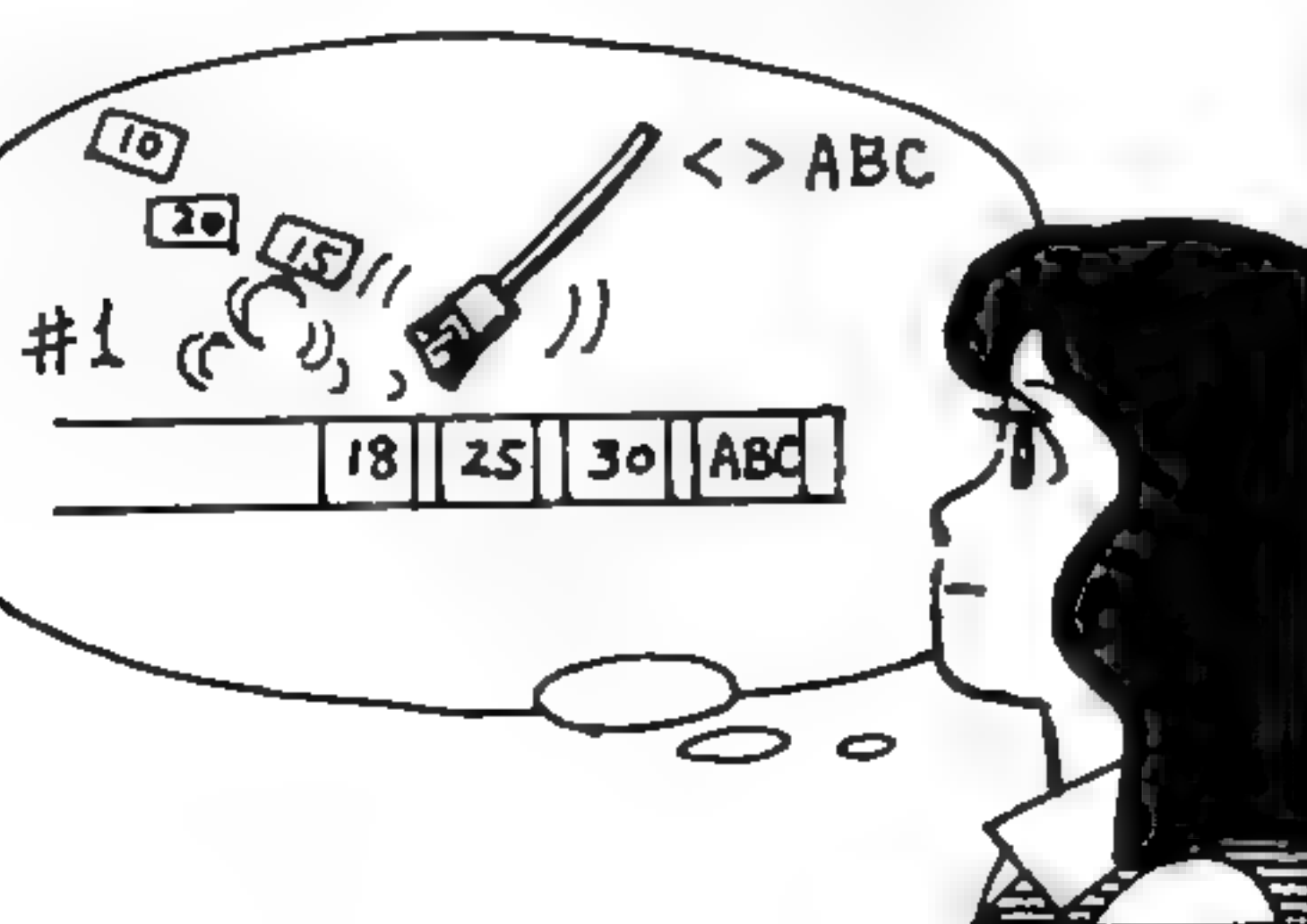
このヘッダーは別にABCでなくてもよいのですが、マニュアルを参考にしましたのでそのまま使いました。

320: LPRINT "ABC"

途中でプログラムをキャンセルした時などバッファの中にデータが残ることがあるのよね。



バッファの中身が何かはわかりませんからRIGHT\$(Y\$, 3) <> 'ABC'は、バッファの中身を掃除することになります。



通信用バッファはFIRST IN, FIRST OUTです。先に入ったデータが先に取り出されます。

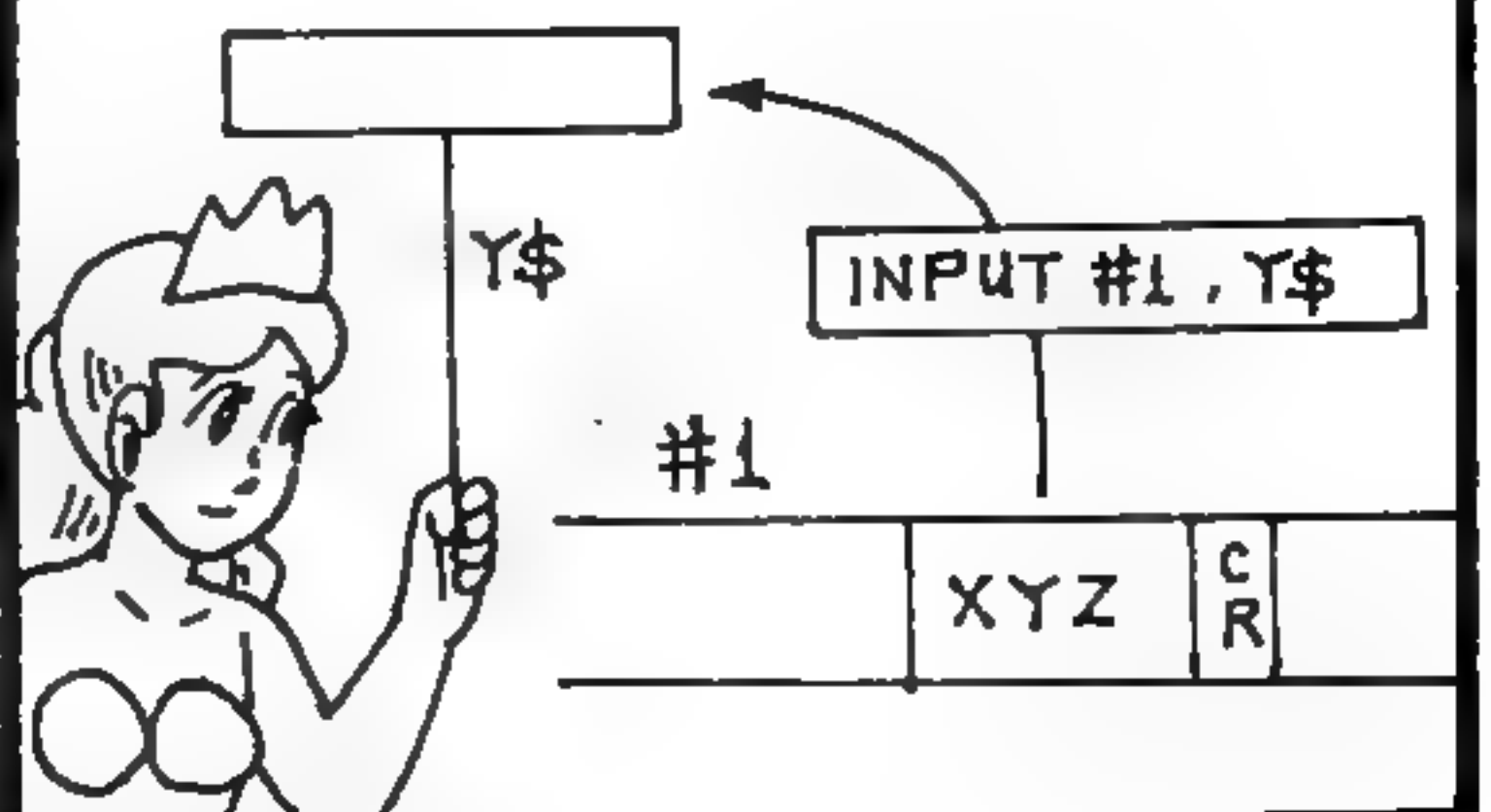
これが次のINPUT #1, Y\$でY\$に取り出されるデータです。

#1

20	C	18	C	15	C
----	---	----	---	----	---

次のデータが入ると以前のデータは前にシフトされます。

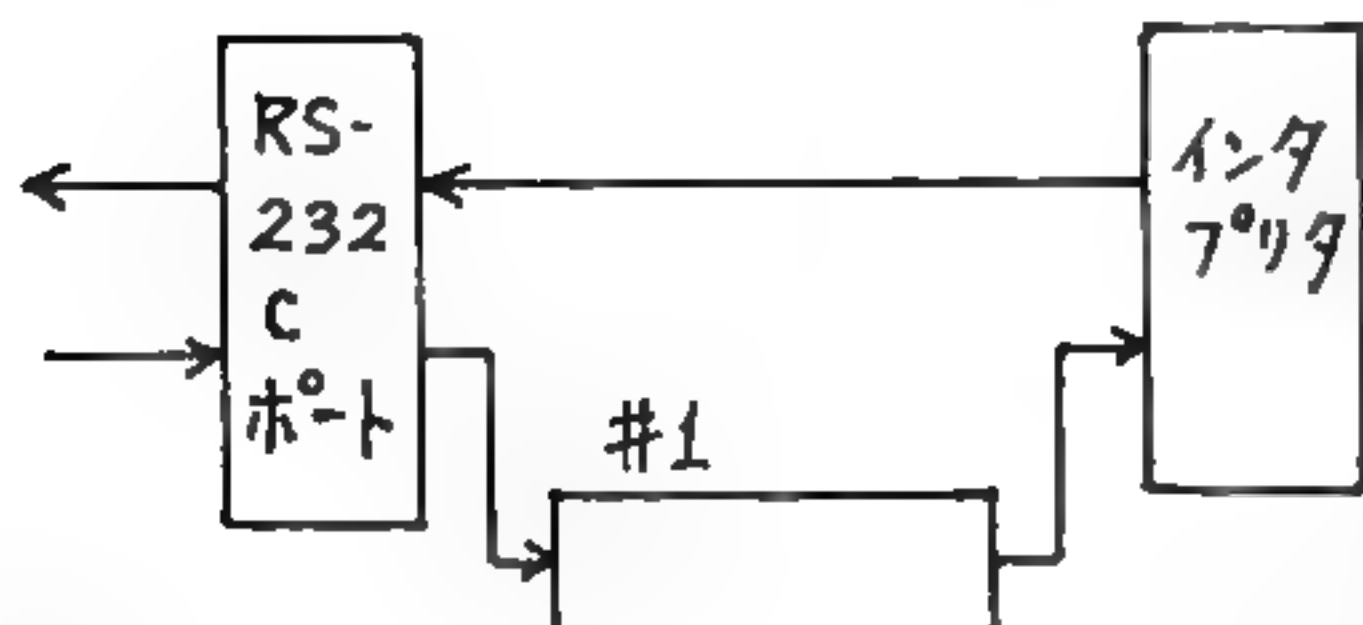
まず、INPUT #1, Y\$ではバッファから先頭のデータ(最初のCRまで)を取り出し変数Y\$に入れます。取り出されたデータはバッファから消えます。



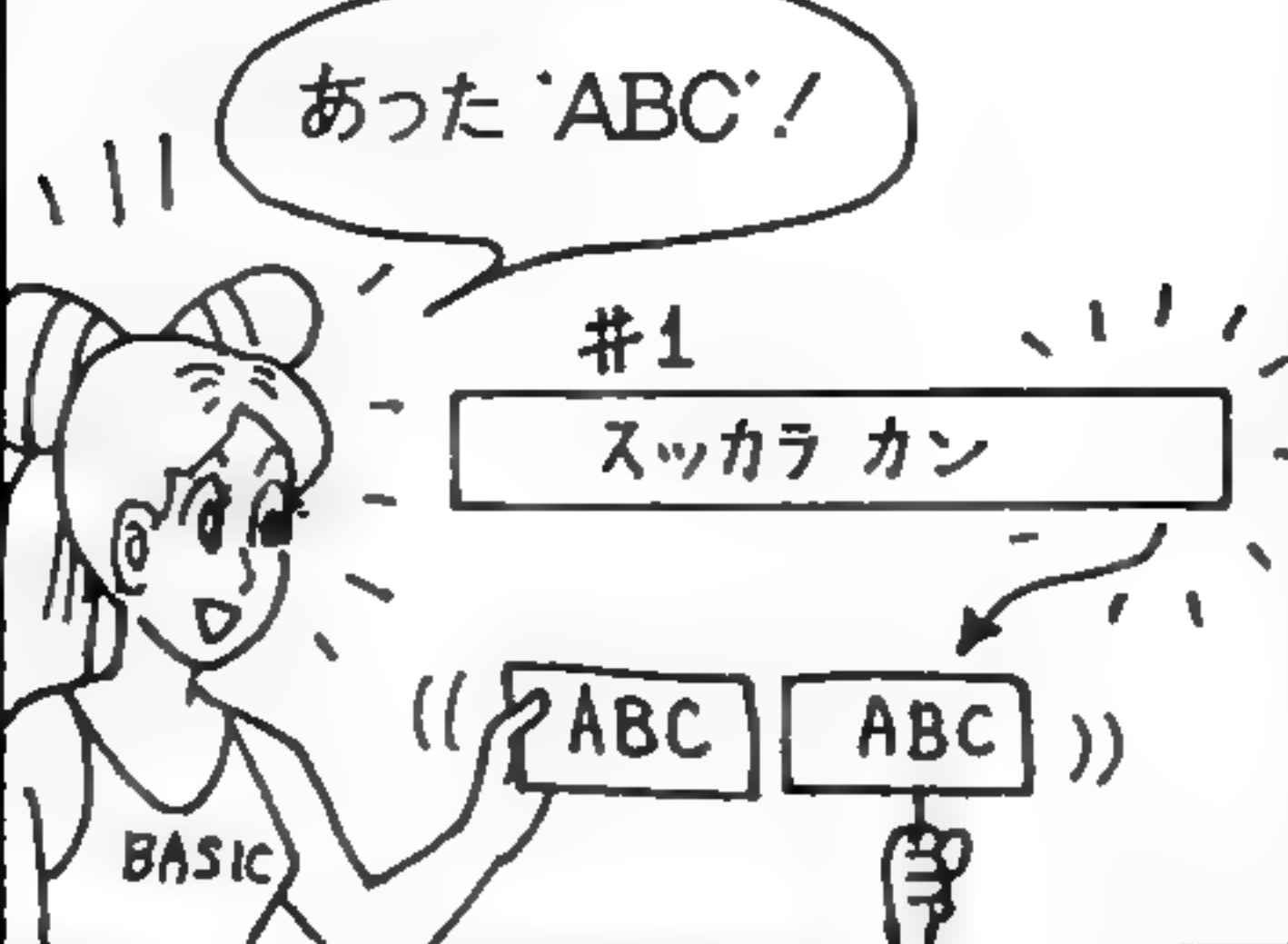


そこでIF800-30からも、キーワードとして'ABC'を返してやります。

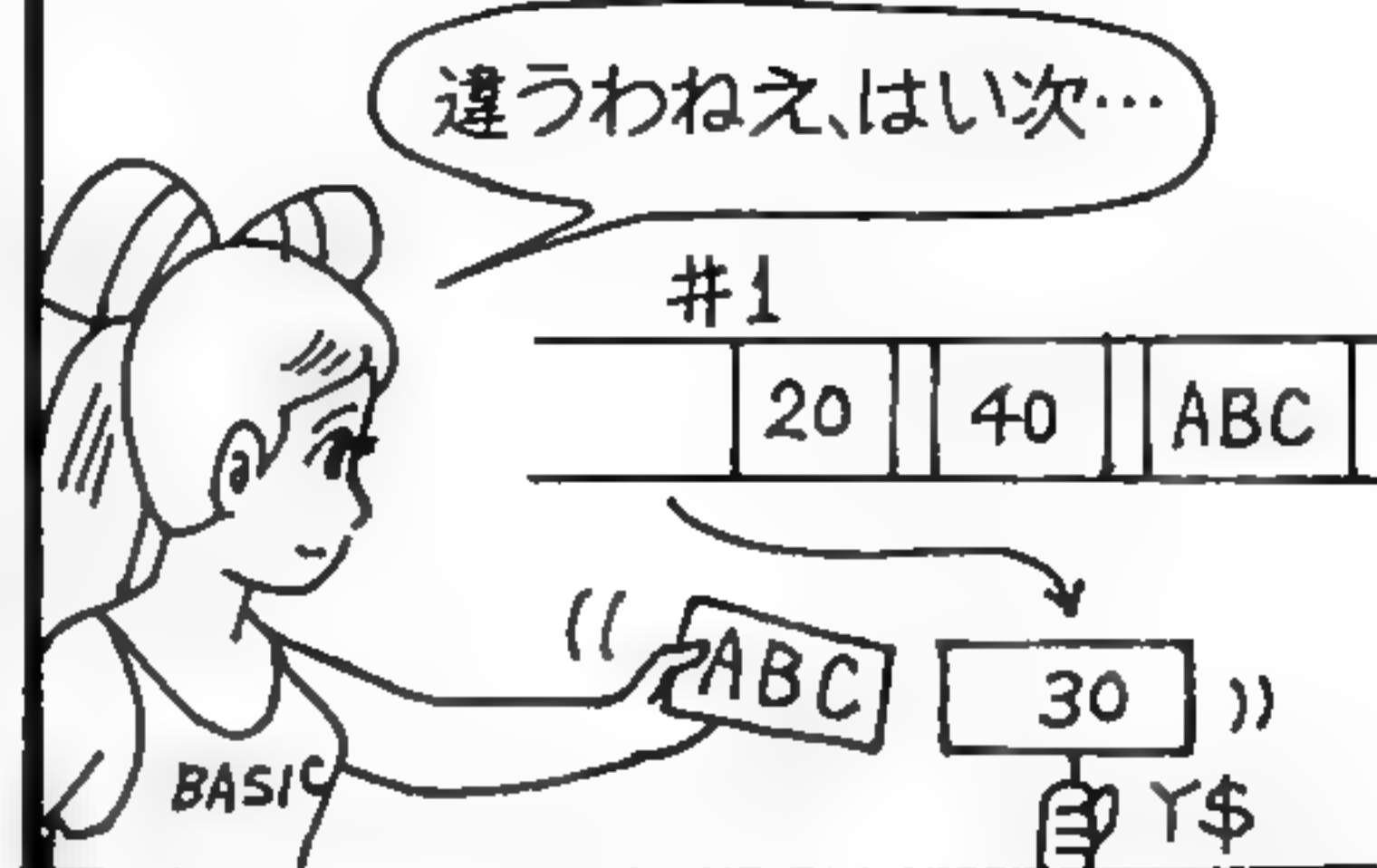
```
80: PRINT #1, "ABC"
```



ポケコン側からはヘッダー'ABC'が送られただけです。バッファから'ABC'が取り出された時点でバッファの中はカラっぽになってしまいます。



それはY\$に'ABC'が入るまでバッファからデータを取り出しては捨てているからです。つまりヘッダー'ABC'以前のバッファの中身は不要というわけです。



これで送受信側とも準備完了というわけです。RS-232Cには先に説明した制御信号がありますが操作法がわからないし、あてになりません。ソフトで工夫した方が確かです。



ポケコン側ではヘッダー'ABC'を送ったあと、SETDEV KIとして、INPUT Y\$に'ABC'が来るのを待っています。この場合もポケコン側のバッファが掃除されることになっています。

```
330: SETDEV KI: INPUT Y$: PRINT Y$:
    IF RIGHT$(Y$, 3) <> "ABC" THEN 330
```



その次が配列の中身です。一足先に送ったNを使って配列を宣言する必要があります。

```
430: FOR J=0 TO 2
440: FOR I=1 TO N
450: P=A2(I,J)
460: LPRINT P
470: NEXT I
480: NEXT J
```

どうして数値変数と文字変数を分けたかというと、INPUT #1のあとに両方混合したものをつけると表示がおかしくなったからです。

```
ポケコン
LPRINT C$, F, G
↓
IF800-30
INPUT #1, C$, F, G
```

次にポケコン側は、まず数値変数の中の日データをまとめて出力します。次に文字変数の中身を出力します。

```
400: LPRINT C, D, E, N, B,
    R, S, U, F, G
420: LPRINT C$
```

IF800-30側でもポケコン側が送り出したのと同じ順序でバッファから各変数に代入してやらなければなりません。

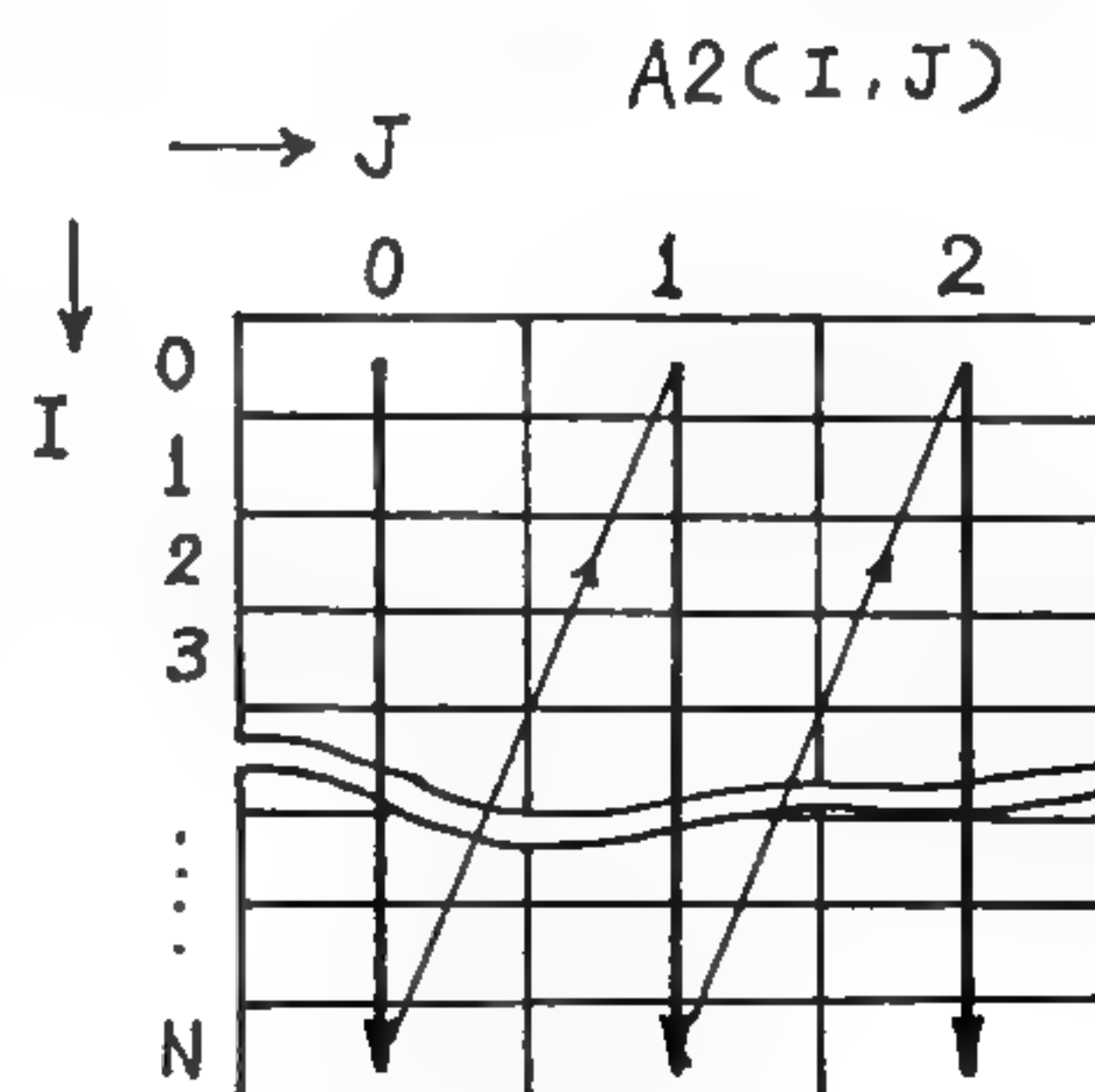
```
90 INPUT #1, C, D, E, N, B, R, S, U, F, G
120 INPUT #1, C$
150 DIM A2(N, 2): Z=1
160 FOR J=0 TO 2
170 FOR I=1 TO N
180 INPUT #1, A2(I,J)
190 NEXT I
200 NEXT J
```

どこに何が入ってくるかさえわかっていれば別に同じ変数名を使わなくてもいいんですよ。

IF800側では配列に直接読み込みます。



データはこの順序で出ていきます。





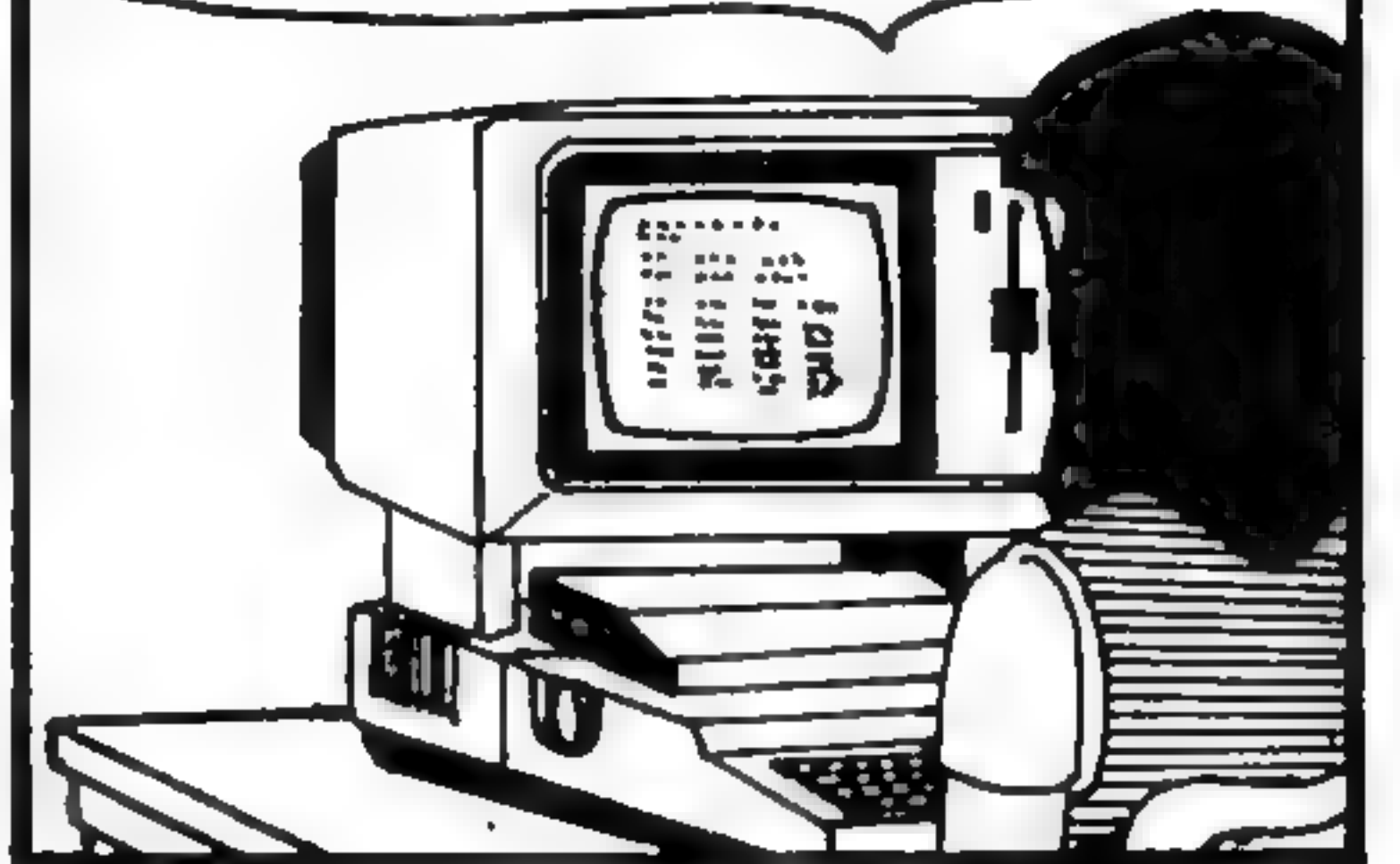
では、このシステムを動かしてみよう。具体的な操作を教えてもらうことは慣れることにつながります。習ったことだけでは自分のものになりません。



ポケコンがカセットからデータを読んでいる間に何か別の仕事ができないか...というようなことは単発的なプログラムでは期待しない方がいいですね。

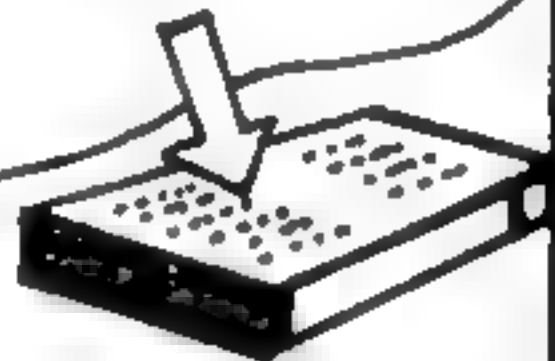


文250は表示のための時間かせぎに使っています。文40に戻るとCLSでクリアされてしまうからです。ポケコンから次のデータが送られるまで3分程度ありますので、2分ぐらいこうしていてもいいのです。

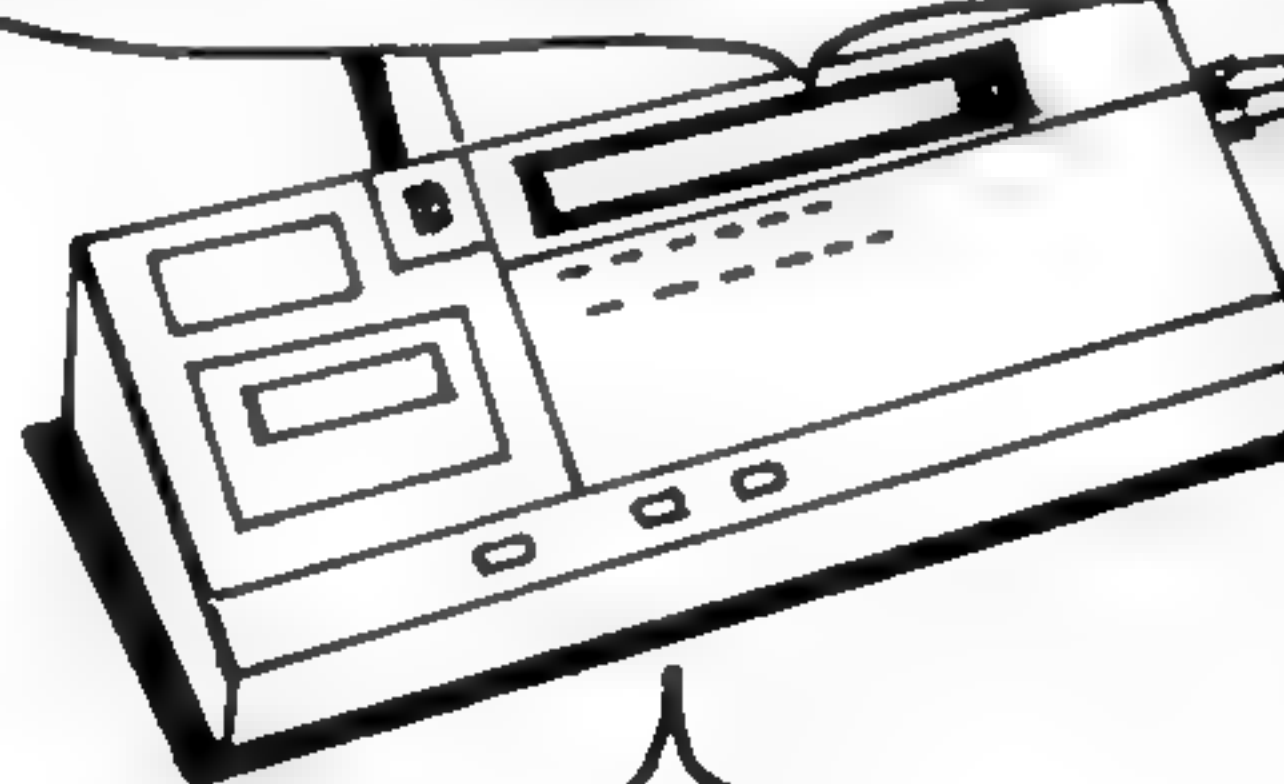


プログラムをセーブする時にも、ヘッダーをつけておくと便利です。カセットにはこのヘッダーとプログラム名とカウンターの値を記入しておきます。

ヘッダー	プログラム名	カウンタ
A	PC1500 TO IF800-30 BY RS-232C (データカセットイン)	005 ~0013



プログラムを入れたカセットをセットしPLAY(再生)スイッチをONにしておきます。



ポケコン側のリモートスイッチをONにしておかないとスイッチが回ってしまいます。

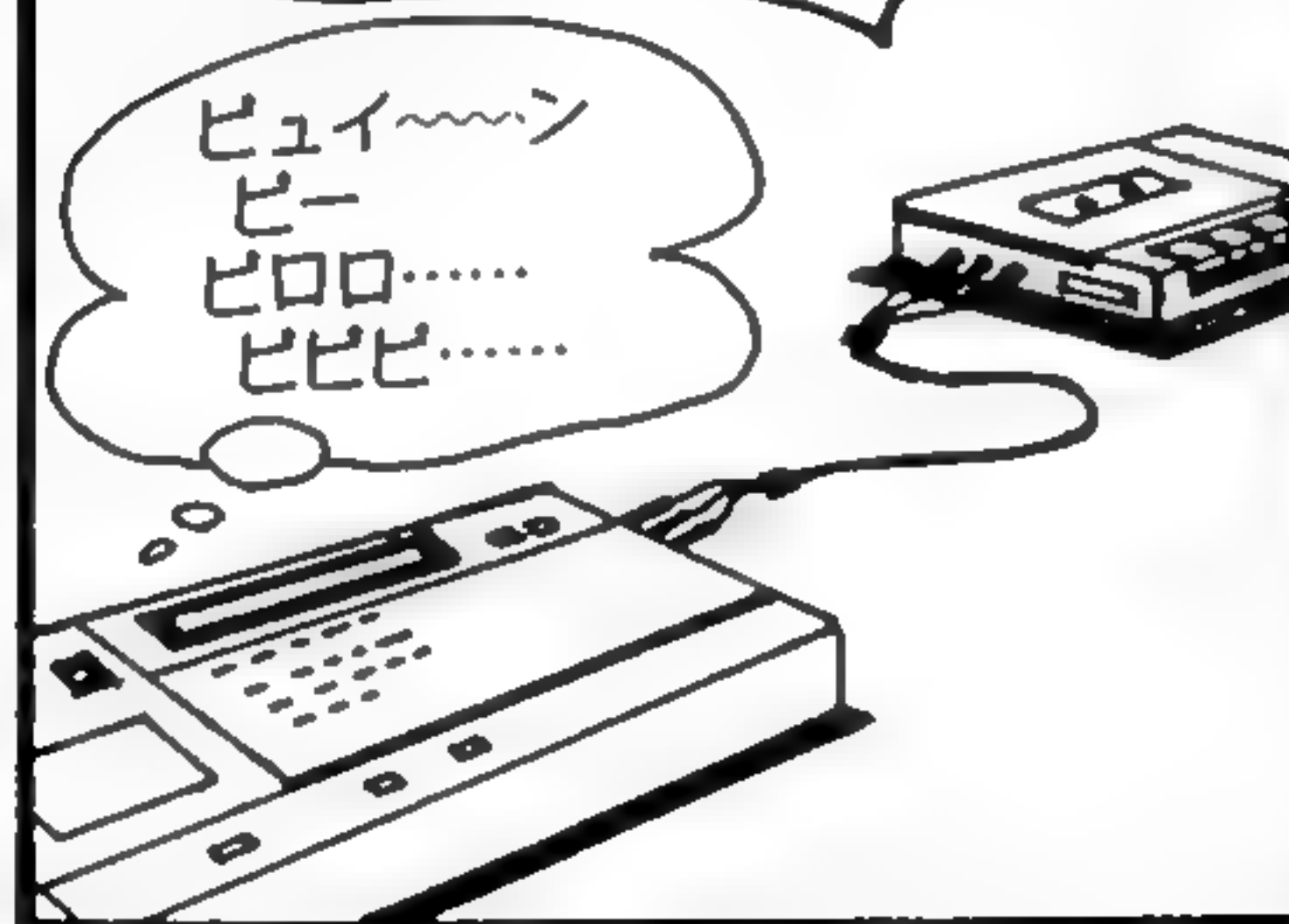
まずテープレコーダーです。ボリュームをMax(最高)にしておくのがポイントです。また、バッテリーよりもDCアダプターを使った方が確実でした。



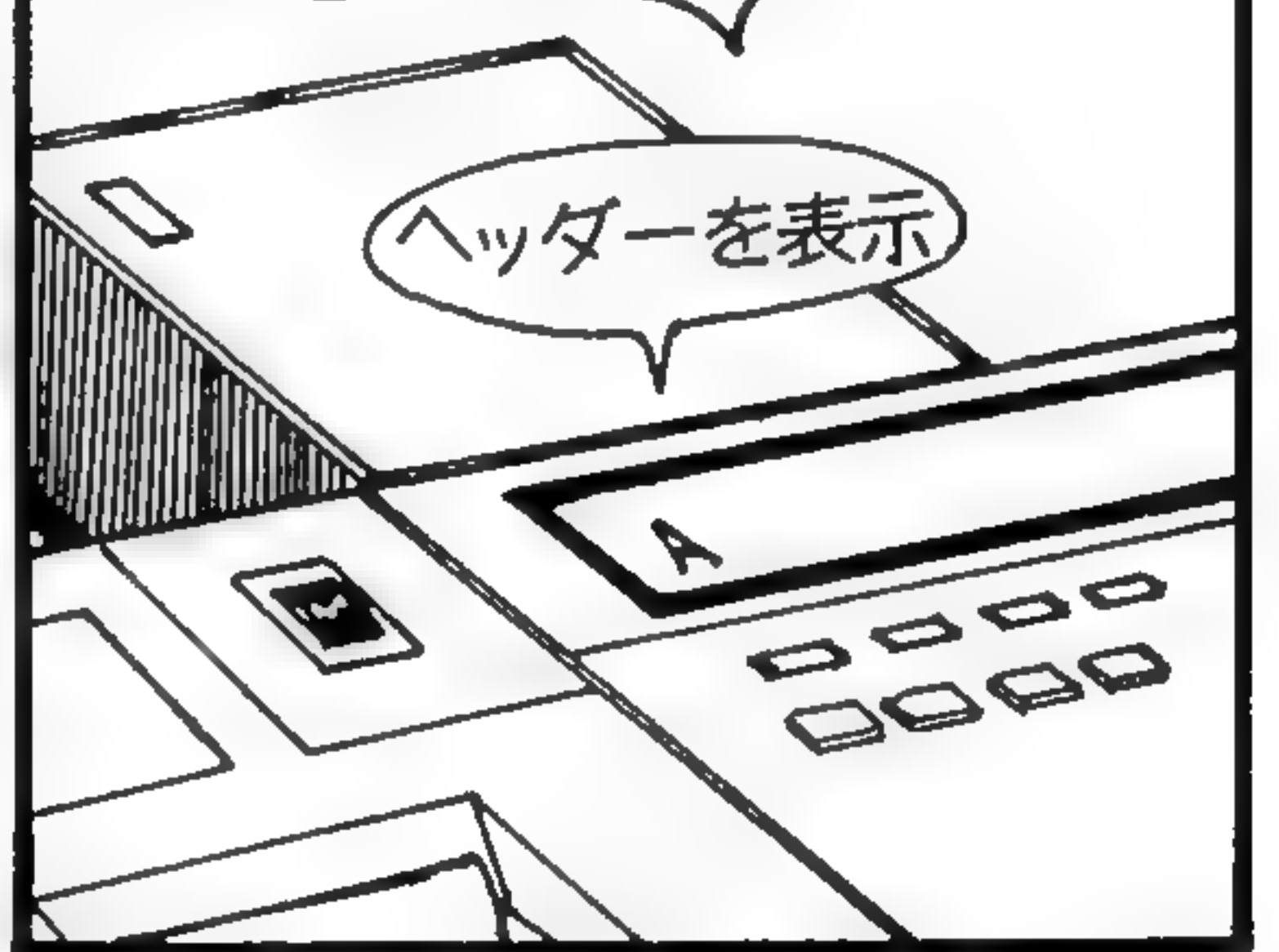
プログラムロードが完了すると、表示は>(フロンスト)に戻りテープの回転が止まります。PLAYスイッチをOFFにします。



この時ポケコンは再生音を出します。この音はBEEP OFF ENTERで消すことができ、BEEP ON ENTERで出すことができます。



CLOAD'A ENTERでテープが回ります。ポケコンはヘッダーが見つかる则表示します。



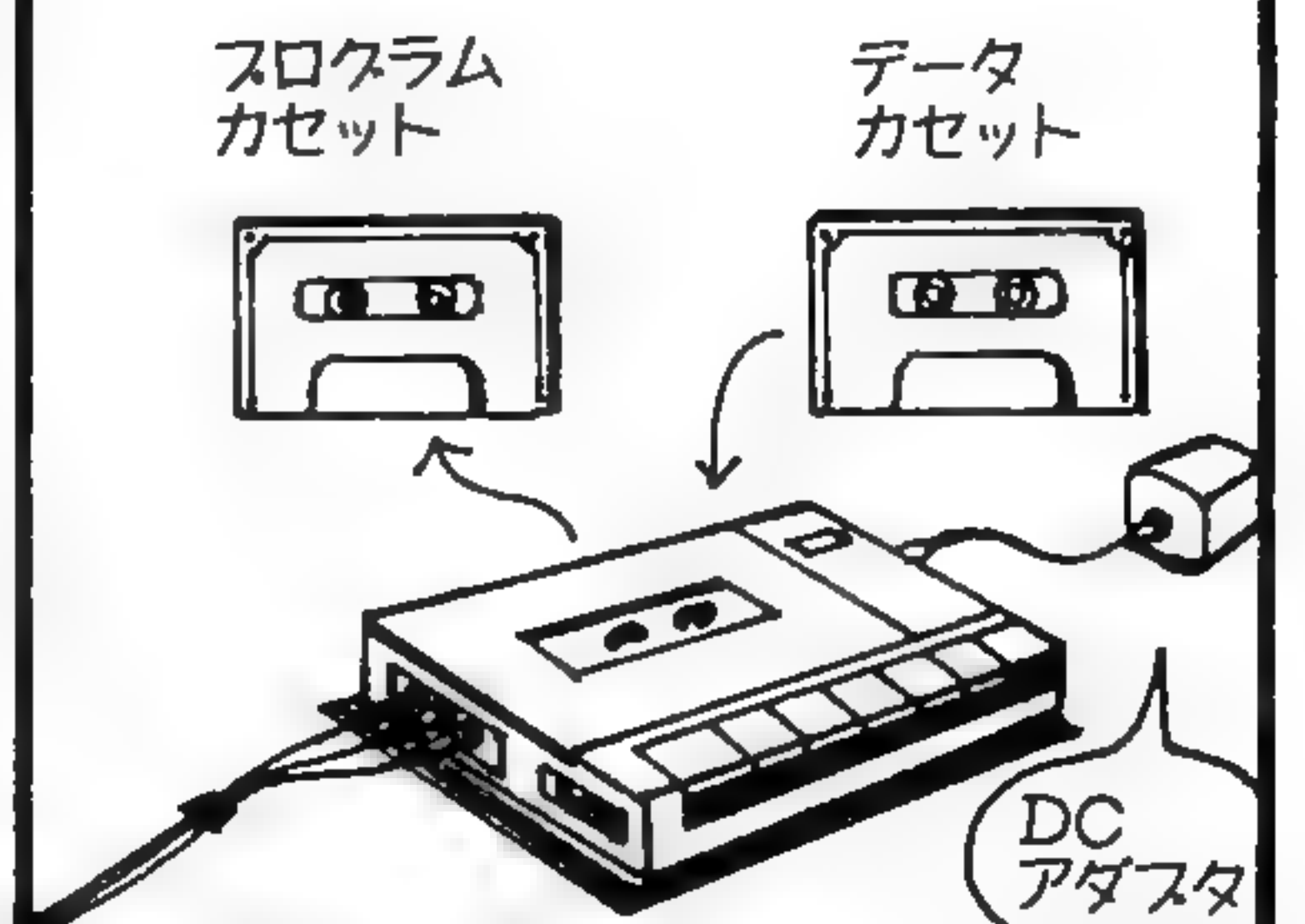
RUNすると、まず'ABC'が送出されますがポケコン側が起動していないのでたれ流しとなります。制御信号も効いていないことがわかりました。次のメッセージを表示してポケコンからのデータ待ちとなります。

PC1500 TO IF800-30 BY RS-232C

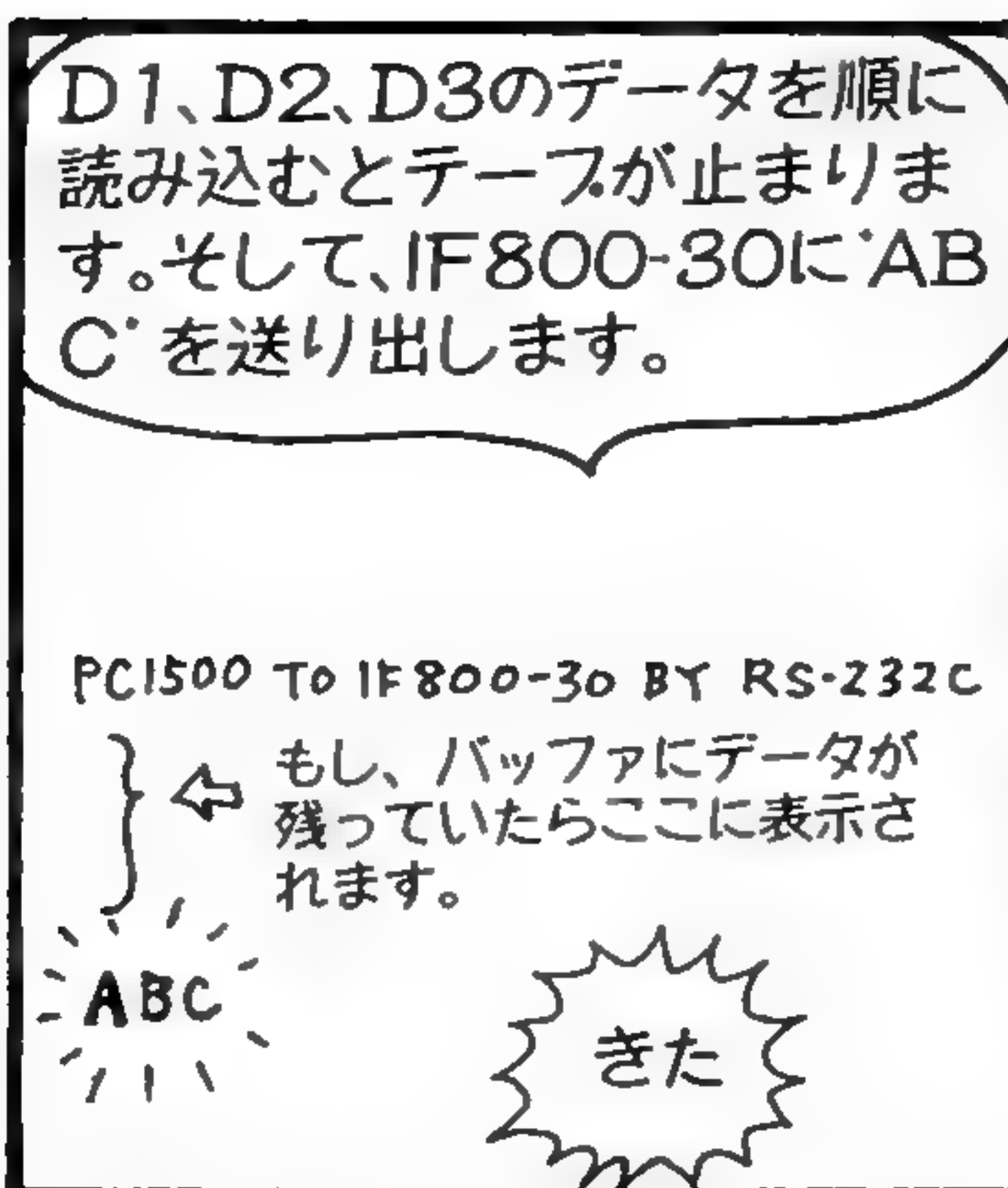
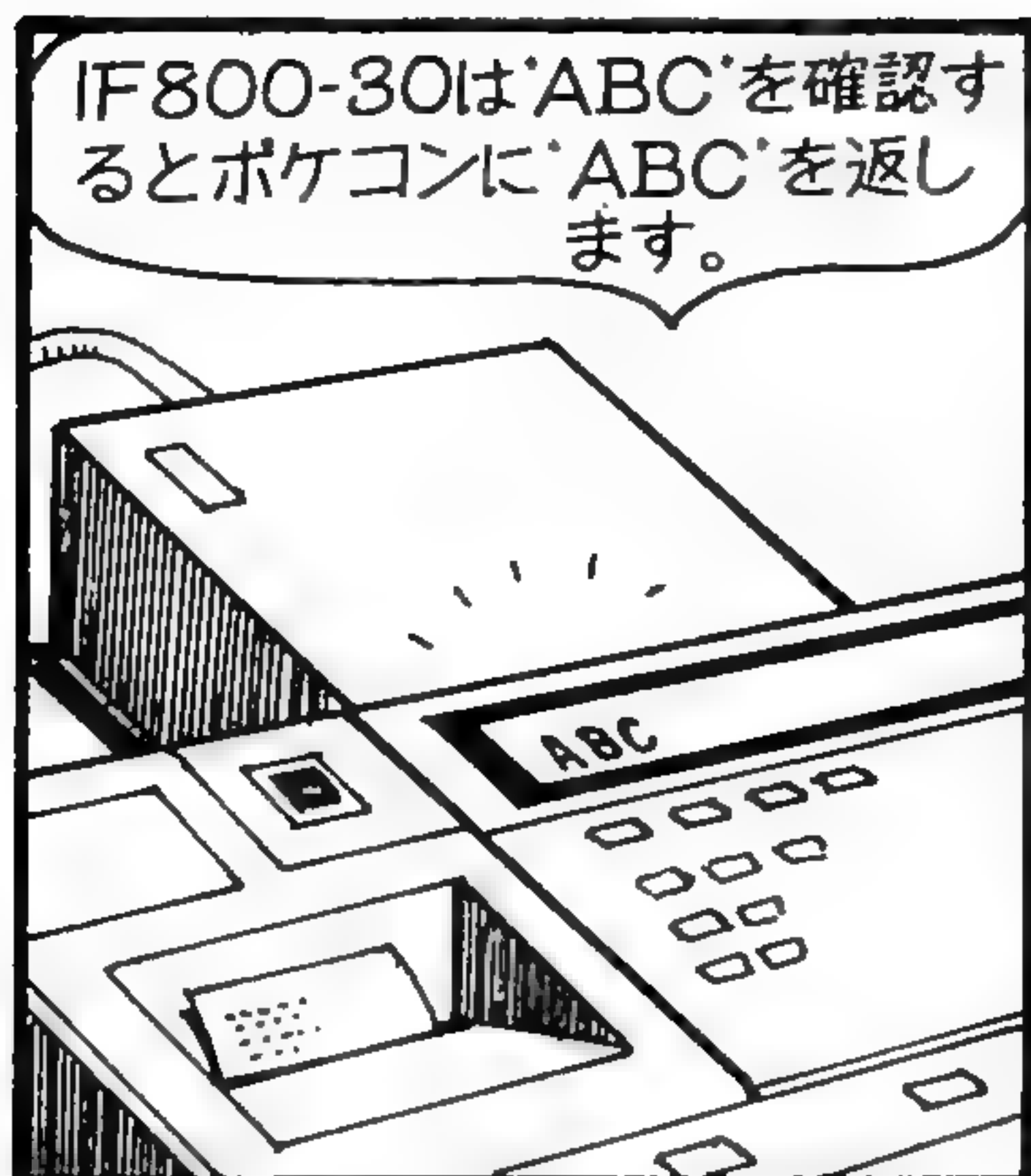
次はIF800-30側の起動です。プログラムはRS-232C1でセーブしておきますのでこれをロードします。

LOAD "RS-232C1"  
NOT FOUND  
LOAD "RS232C1"  
OK

次にカセットをデータが入ったものに換えてセットします。PLAYスイッチをONにします。



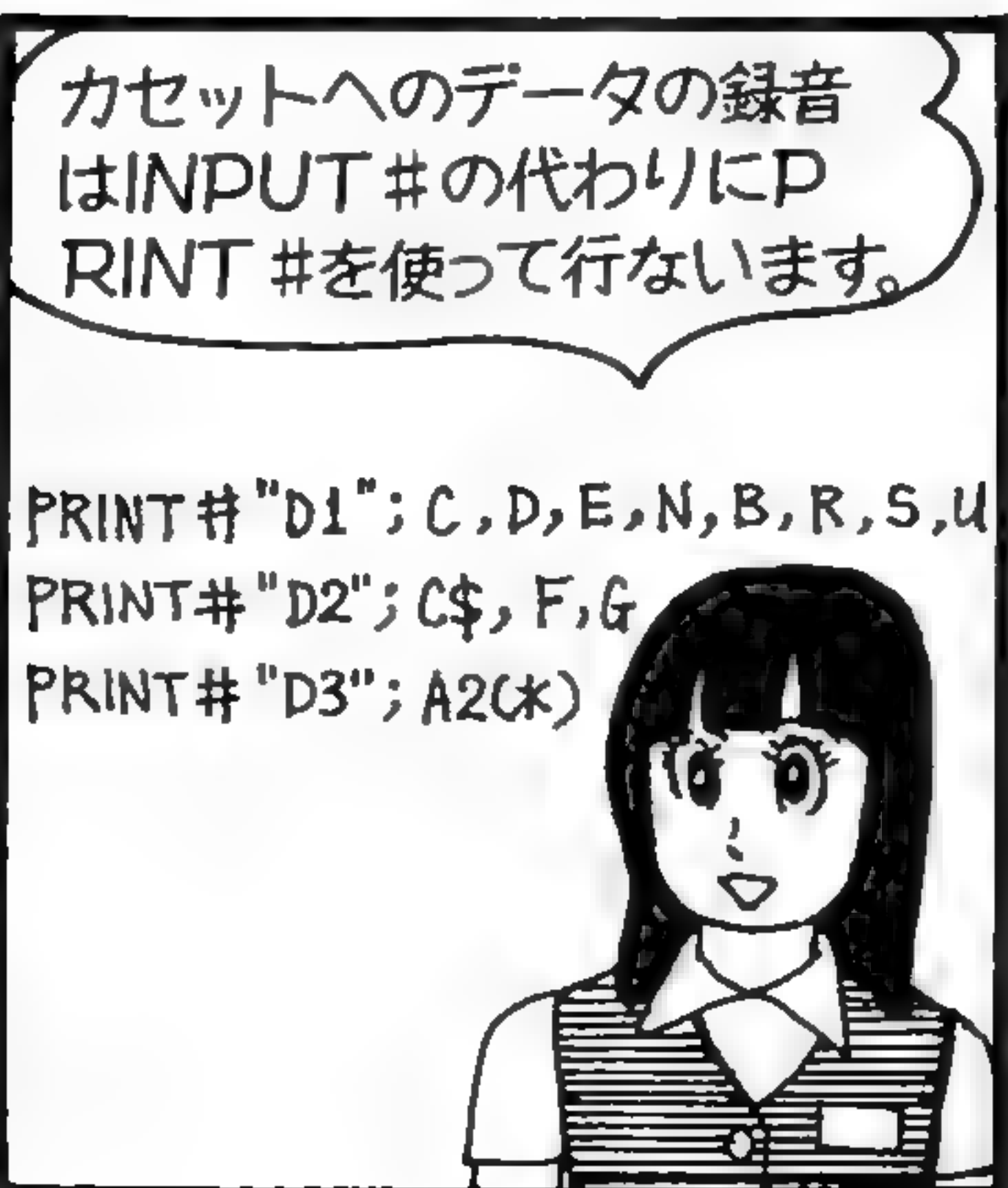
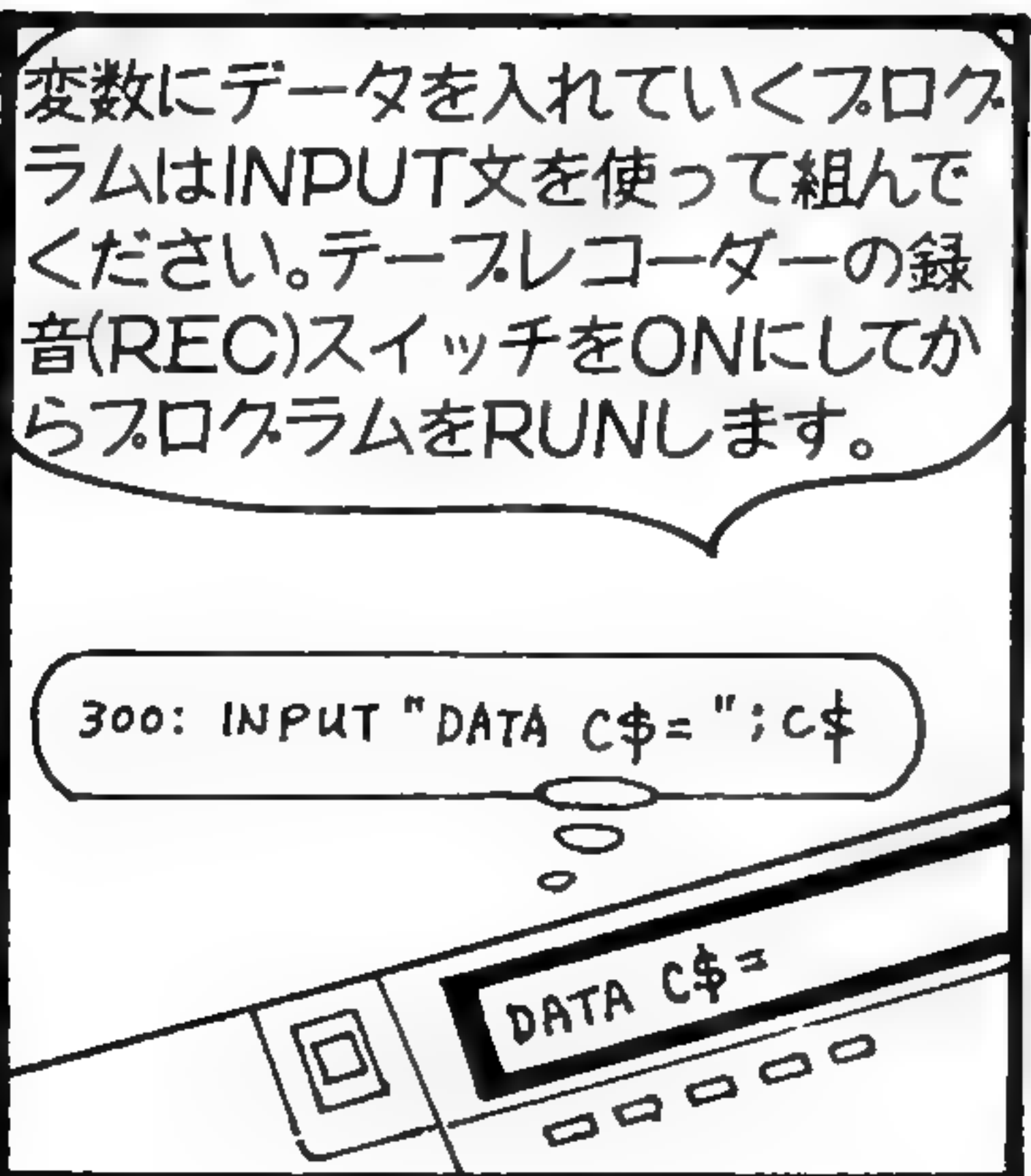
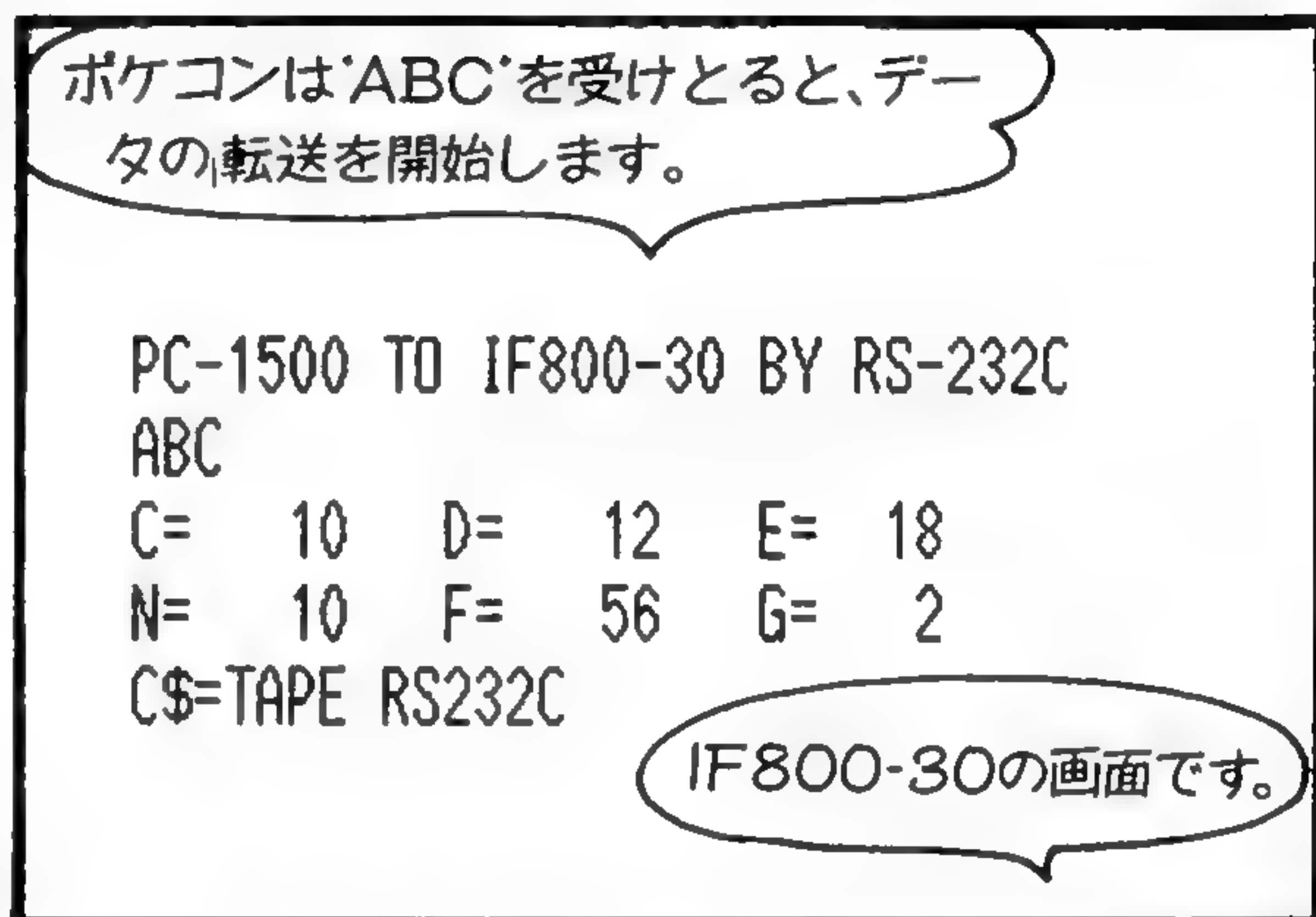




PC-1500 TO IF800-30 BY RS-232C

ABC

C=	10	D=	12	E=	18
N=	10	F=	56	G=	2
C\$=TAPE RS232C					
I	J=0	J=1	J=2		
1	10	30	15		
2	12	10	10		
3	25	15	5		
4	14	18	1		
5	23	25	9		
6	10	16	30		
7	15	10	26		
8	25	10	25		
9	17	15	14		
10	25	16	8		





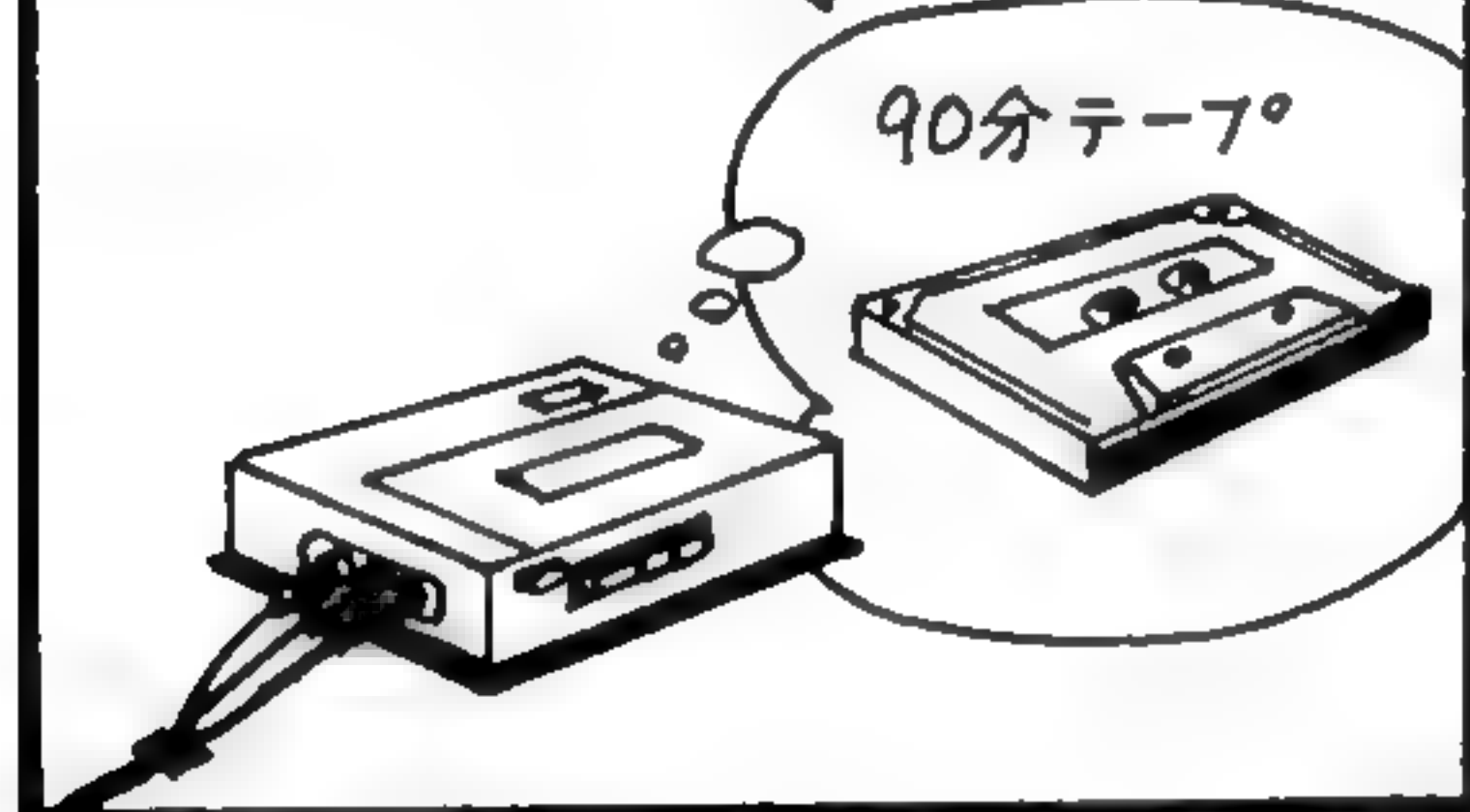
このプログラムで覚えておきたいのは文250と225でデータの訂正機能を持たせていることです。この場合99をインプットすることによってひとつ前に戻れます。この99はこのプログラムではあるはずのない数字なのです。

```
220: IF A2(I,J) <> 99
      THEN 230
225: I = I - 2
```

参考のためにインプットしたデータをテープに録音するプログラムを作ってみました。1レコードのフォーマットは209ページのプログラムに合わせています。



プログラムをRUNする前に新しいテープをセットします。またカウンタをゼロにします。PC-1500とテープレコーダを接続しリモートスイッチをONにしてから録音(REC)スイッチを入れます。テープは少し回しておきます。

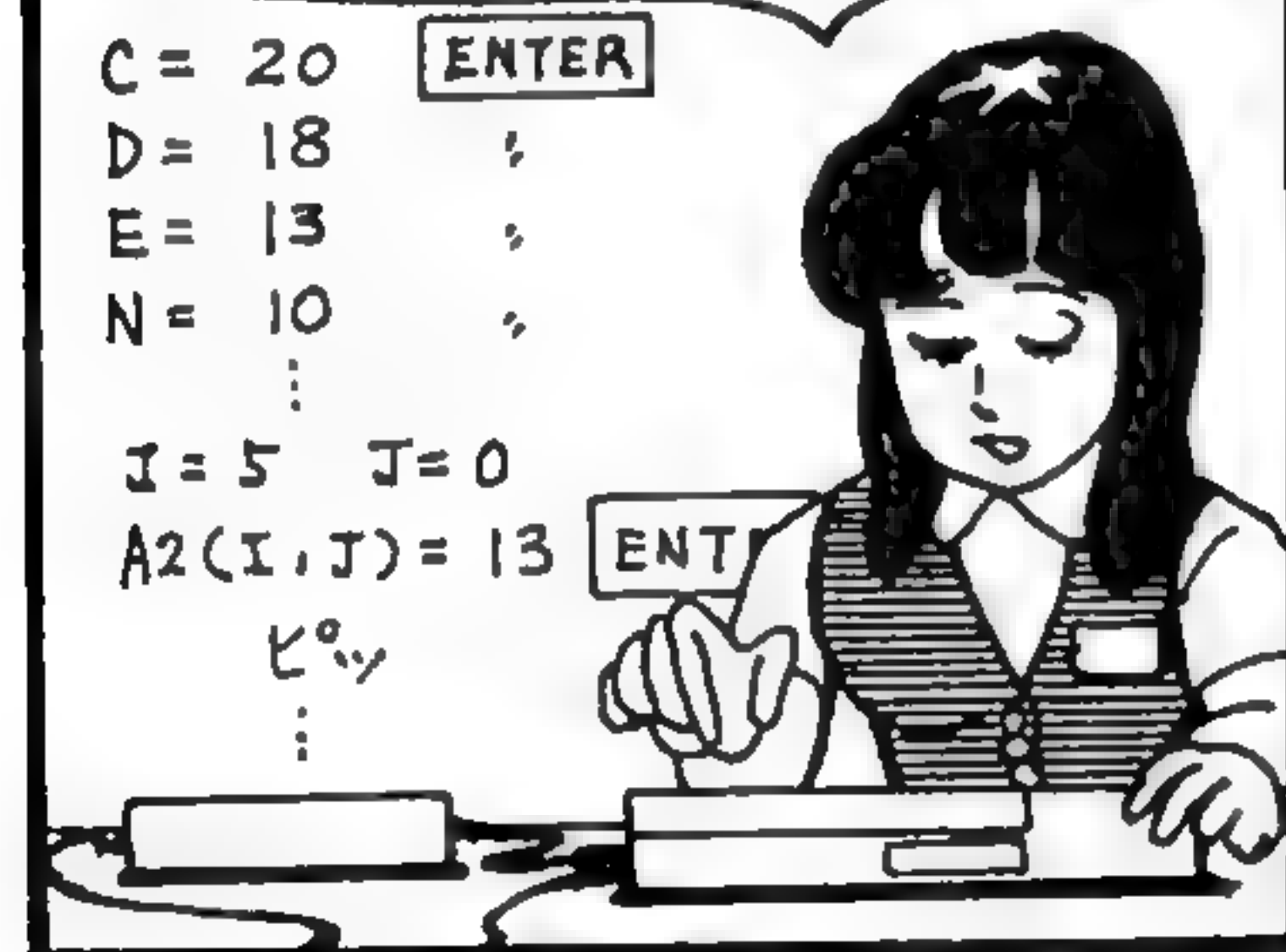


文270のA2(\*)は配列A2(I, J)全部の要素つまりN=10なら33個の変数を示します。特定の要素を抜き出して録音したい時は209ページのプログラムの文450のような工夫が必要です。

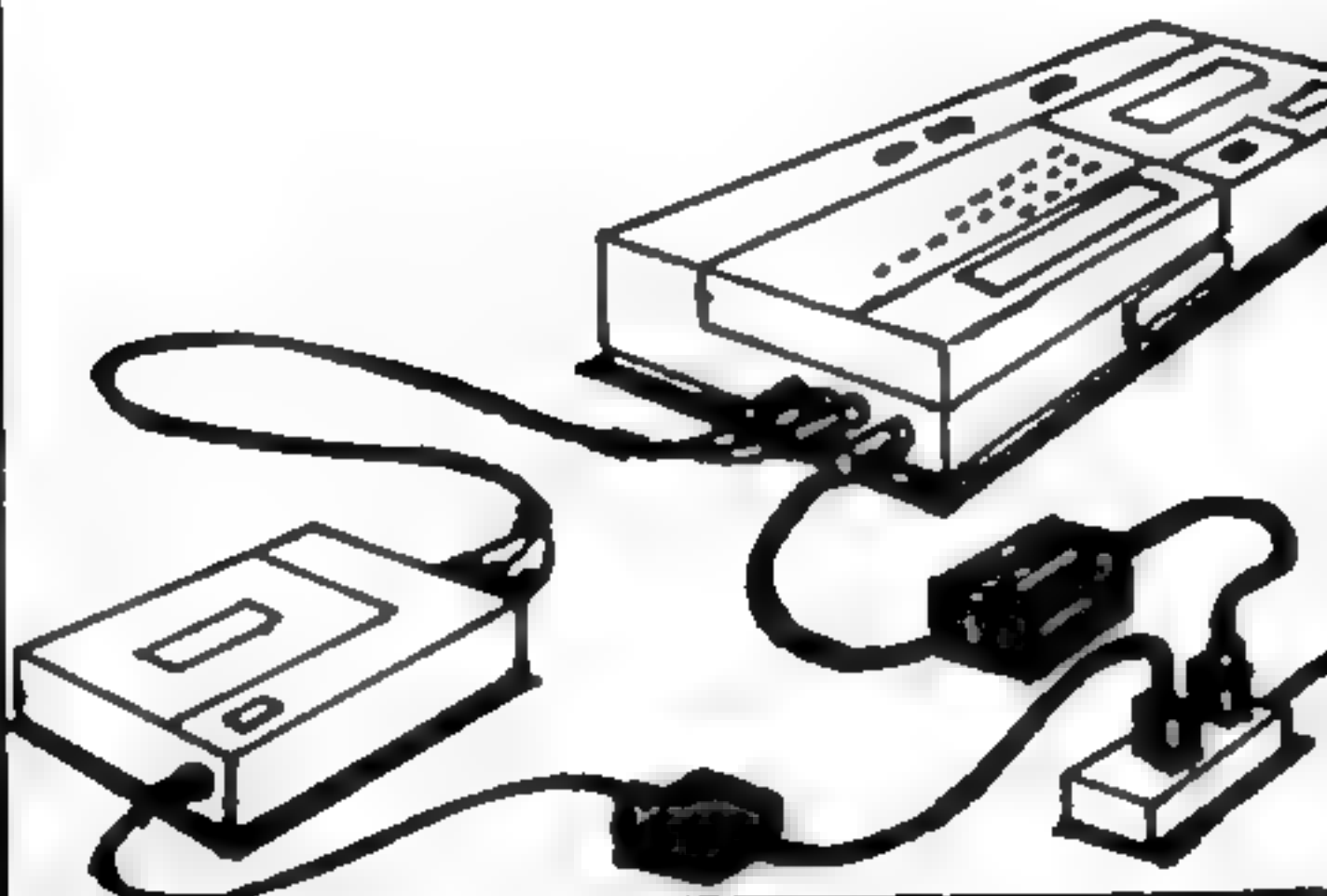
	I	J	0	1	2
A(*)	0				
1					
2					
3					
4					
5					
...					

次々に表示されるガイダンスに従いデータを入力します。配列へのインプットの時はBEEP文でビップと鳴るようになっています。

```
C = 20  ENTER
D = 18  ,
E = 13  ,
N = 10  ,
      :
J = 5   J = 0
A2(I, J) = 13  ENTER
      ビップ
      :
```



RUNすると、TAPE SET OK? と表示されます。OKであればENTERキーを押します。



離れた所にあるパソコンとつなぐ時は音響カプラーと電話を使えば簡単にできるのです。

とにかく、使ってみること！  
RS-232C



Nに1をインプットするとビップビップビップと5回鳴ってプログラム終了です。これで例示のプログラムのインプット用テープができました。

N=1はあり得ない数字なのでエンド・マークとして使いました。



インプットが完了するとテープが回転を始めます。そして録音が完了するとテープは停止し、Nが1でない時は文40に戻ります。

ビュイ  
イン  
ビロビロ



```
10: REM AN EXAMPLE
    OF TAPE INPUT
20: Z=0
30: WAIT : PRINT "T
    APE SET(REC)
    OK?"
40: INPUT "C="; C
50: INPUT "D="; D
60: INPUT "E="; E
70: INPUT "N="; N:
    REM TAPE END M
    ARK N=1
80: INPUT "B="; B
90: INPUT "R="; R
100: INPUT "S="; S
110: INPUT "U="; U
120: INPUT "F="; F
130: INPUT "G="; G
140: INPUT "C$="; C$
150: IF Z=1 THEN 170
160: DIM A2(N, 2): Z=
    1
170: WAIT 50
180: FOR J=0 TO 2
190: FOR I=1 TO N
200: PRINT "I=";
    USING "###"; I;
    "      J="; J: BEEP
    1
210: INPUT "A2(I, J)
    =" ; A2(I, J)
220: IF A2(I, J) <> 99
    THEN 230
225: I = I - 2
230: NEXT I
240: NEXT J
245: PRINT "TAPE OU
    T"
250: PRINT "#D1"; C,
    D, E, N, B, R, S, U
260: PRINT "#D2"; C$,
    F, G
270: PRINT "#D3"; A2
    (*)
280: IF N <> 1 THEN 40
290: BEEP 5: PRINT "
    END": END
```



## RS-232CのQ&A

\* 近ごろRS-232Cが注目を集めていますが…。

RS-232Cは、パソコン以前からテレックスなどの通信に使われていた規格です。大型コンピュータでは、端末機器との接続は常識になっています。パソコンもその性能が上がるにつれ、単独使用から端末機器と接続して使用することがパソコン活用の次のステップと考えられます。今やBASIC言語で簡単にデータの送受信が行なえるようになっています。あなたのやる気しだいで、すぐにもパソコンを使ったデータ通信が可能なのです。ただパソコンによって少しずつプログラムのしかたと入出力信号の方向が違います。でもそんなことは、次の4つのポイントさえおさえておけば大きな問題ではありません。

(1) 通信条件を一致させること

通信速度  
パリティ  
データ長  
ストップビット

(2) コネクタピンの入出力方向の整合

こちらの出力は相手の入力となり相手の出力はこちらの入力となる通常のパソコンはケーブルの中でクロスさせる。

(3) データ個数の一致

送る側のデータ個数と受け取る側のデータ個数は一致していなければなりません。

受信にエラーが起こるとデータがずれるので、必ずヘッダーを送り、受ける側はこれを確認してからデータを変数に読み込むようにします。

```
10 OPEN 'COM1:6N82' AS 1
20 INPUT #1, A$
```



```
30 PRINT 'ジュシン'
   データ=; A$
```

↑ ↑

この2つはKEYが違うので  
注意してください。

```
40 GOTO 20
```

(4) アスキーコードでの入出力

数値は文字型にして送り、受信してから元に戻す。

```
K$ = STR$(K)
```

↓

```
K = VAL(K$)
```

TOMOKO ✓

CR O K O M O T



NEC  
PC8201

```
10 MAXFILES=2
20 OPEN 'COM:6N82' FOR OUTPUT #1
30 OPEN 'COM:' INPUT #2
40 INPUT 'ソウシン' データ=; A$
50 PRINT #1, A$
60 GOTO 40
```

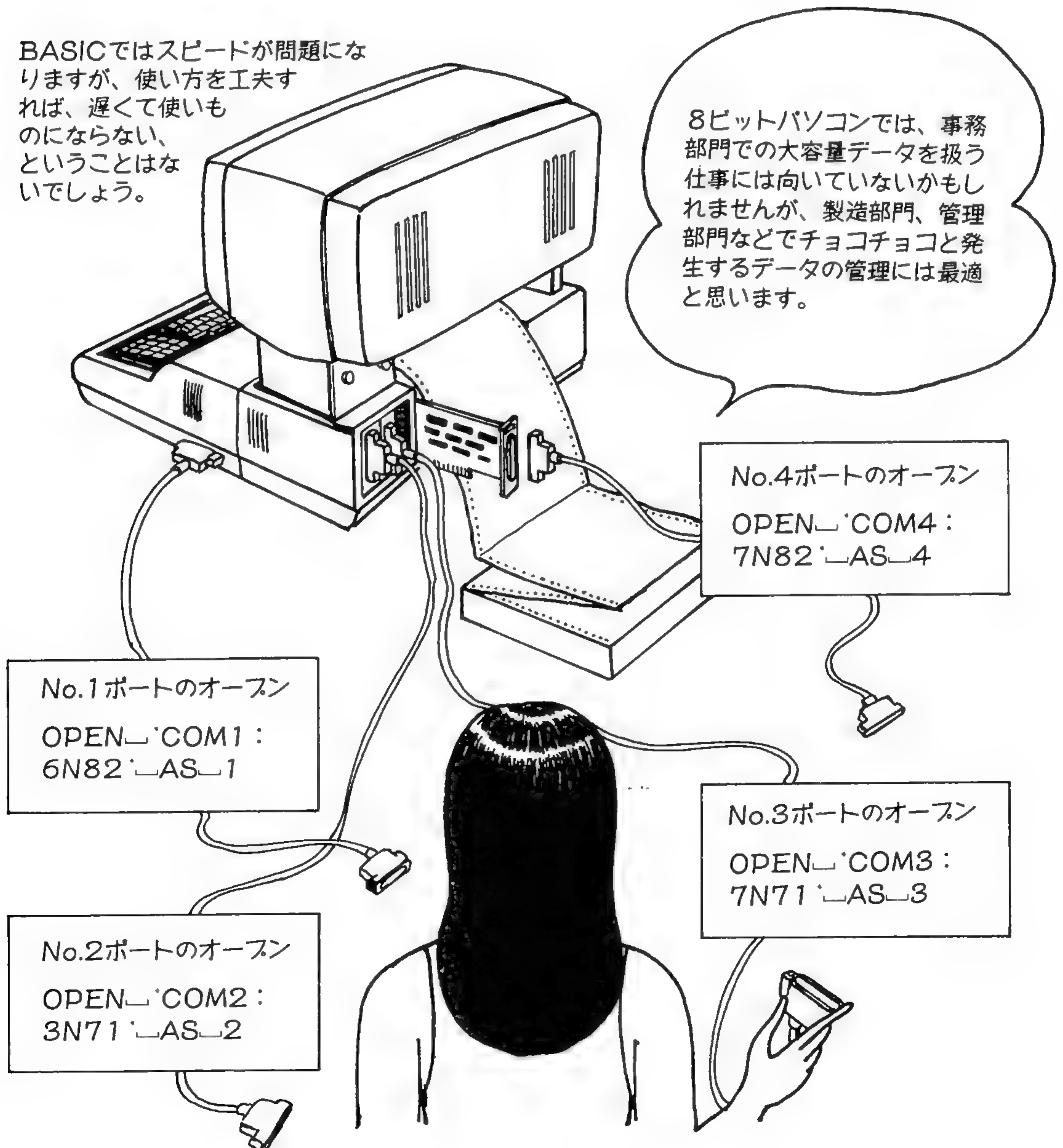


\* BASICでの通信は実用的か？

パソコンが初めて世に出たころのBASICでは、RS-232C通信ポートを扱う命令はサポートされていませんでした。そこで、このポートを使うためにはマシン語プログラムをつくり、これと併用してBASICを使ったのです。このため、通信中はBASICの処理が止まったり、BASIC処理中は通信できないなどのことがありました。

現在では、BASICでこのRS-232Cが使えるようになったほかに、データの受信は割込みルーチンを使ってBASICの処理とは無関係に行なわれ、オープンしたバッファの中にデータが入ります。この図のパソコンでは最大4ポートのRS-232Cが使える、それぞれ独立したバッファにデータを受信できます。通信条件もそれぞれ別個に設定できます。

BASICではスピードが問題になりますが、使い方を工夫すれば、遅くて使いものにならない、ということはないでしょう。





## 【F】データ源から直接入力させる法 (デジタル機器からの出力をBASICで扱うには)

温度にしろ重量にしろ圧力にしろ、制御するためにはまず測定しなければなりません。これら測定器からは、ふつうBCD(2進化10進)の出力がでるようになっています。またバーコードリーダーなどの出力ポートはRS-232Cですから、これらを組合わせると多彩なパソコンデータ処理システムの構築が可能です。

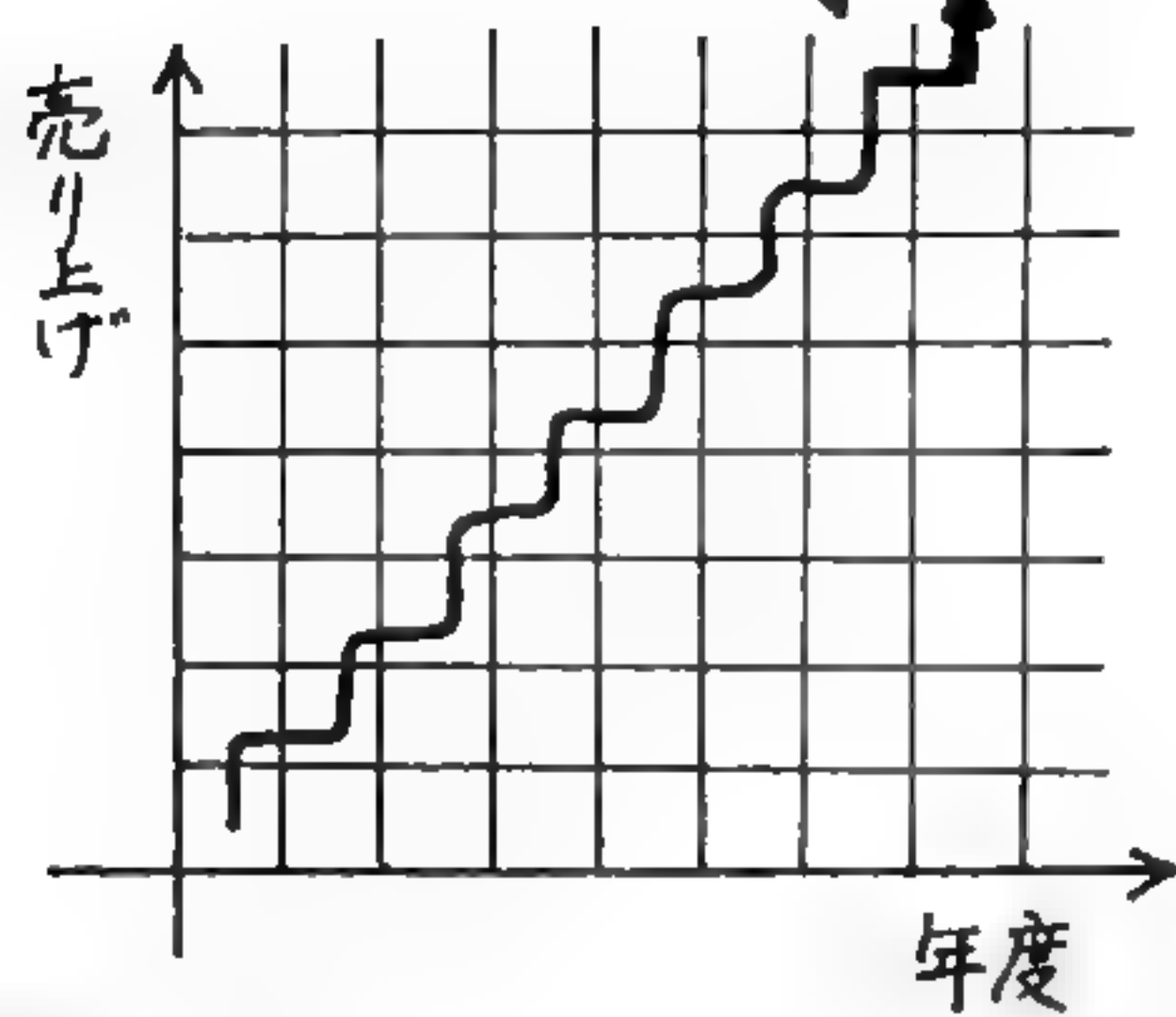
ここでは、BASICでデジタル値をパソコンの変数に取り込むひとつの方法を、具体的な事例でお話します。ここで使ったパソコンはオールRAM型で2つのディスクドライブを持っていますが、他のどんな機種でも応用ができます。

RS-232Cにくらべるとハードがらみのことが多いので、誰でもすぐ応用が効くというわけにはいかないかもしれません。しかし、何でもそうですが全部自分でやる必要はないのです。ひとりでやれることなど知れていることも知っておく必要があります。





電算機発達の原因はこれらデータ処理のニーズがあったからです。



官庁、学校、企業、商店などでは、たくさんのデータの整理、集計なども重要な業務のひとつとなっています。

なんとか損益一覧表

品名	数量	単価	金額	仕入	売上	粗利	経費	利益
1								
2								
3								
4								
5								
6								
7								

## ①工場のハカリとマイコンをつないでみたら



電算機やパソコン導入により事務の自動化いわゆるOAが急速に発達しています。



パソコンは電算機に比べると低価格であり、単能機として使っても問題にならないくらいです。



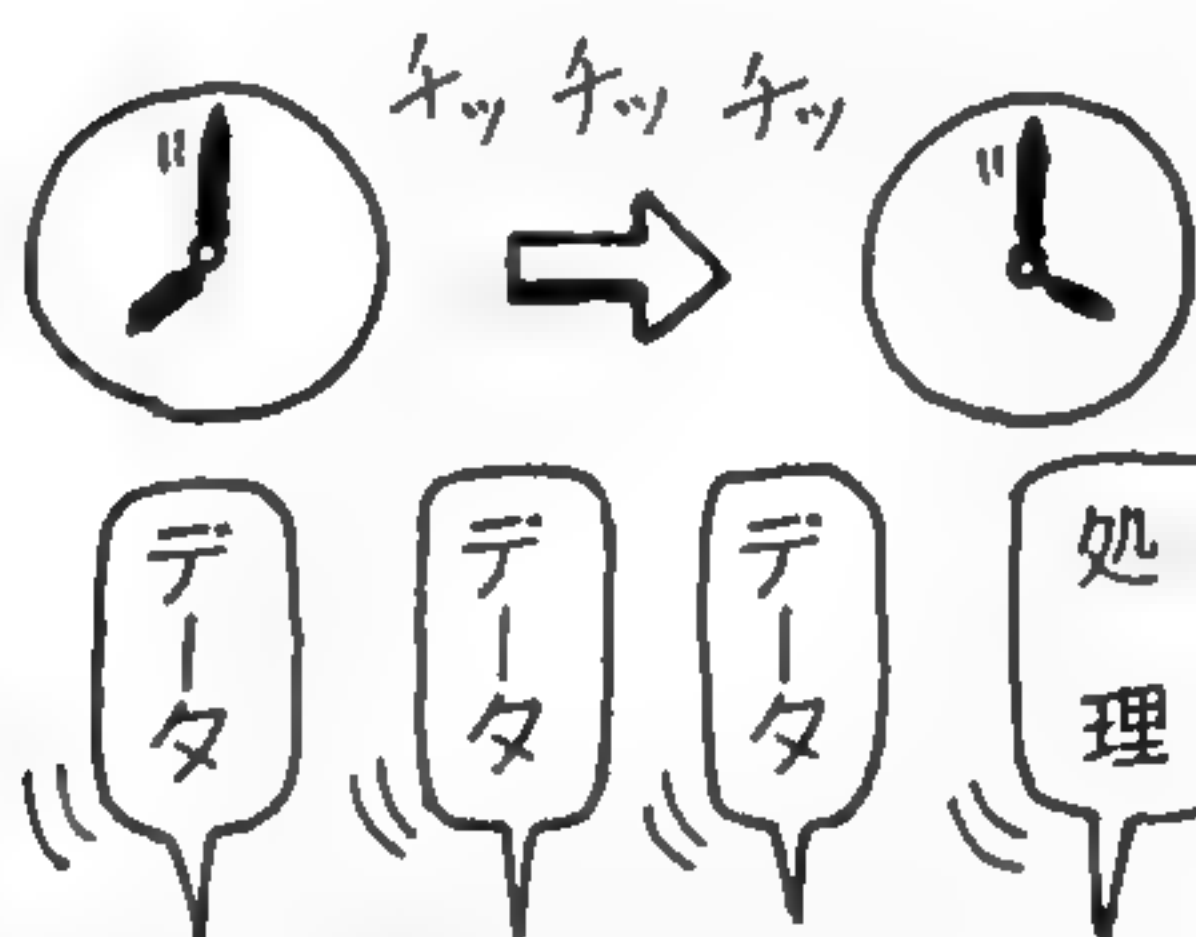
今、パソコンが出現し、その能力は電算機に近づきつつあります。



プログラム処理の都合上これらのデータはある程度まとめて処理されますが、これがバッチ処理です。



多くの場合データの発生からコンピュータへのインプットまでには時間的ズレがあります。



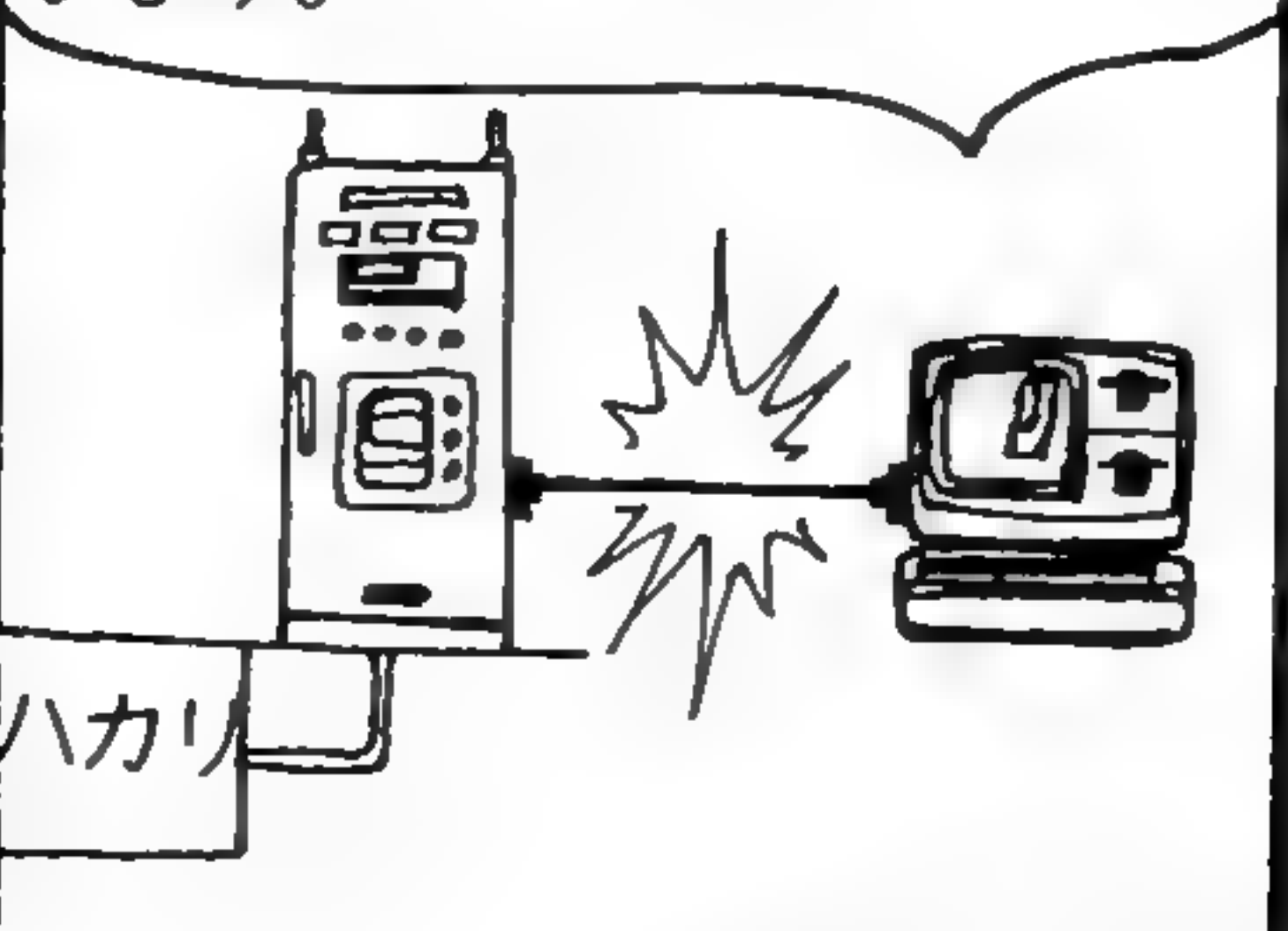
しかし、データの整理、集計などが自動化されてみると、今後はそのデータのインプット業務が目立つようになります。



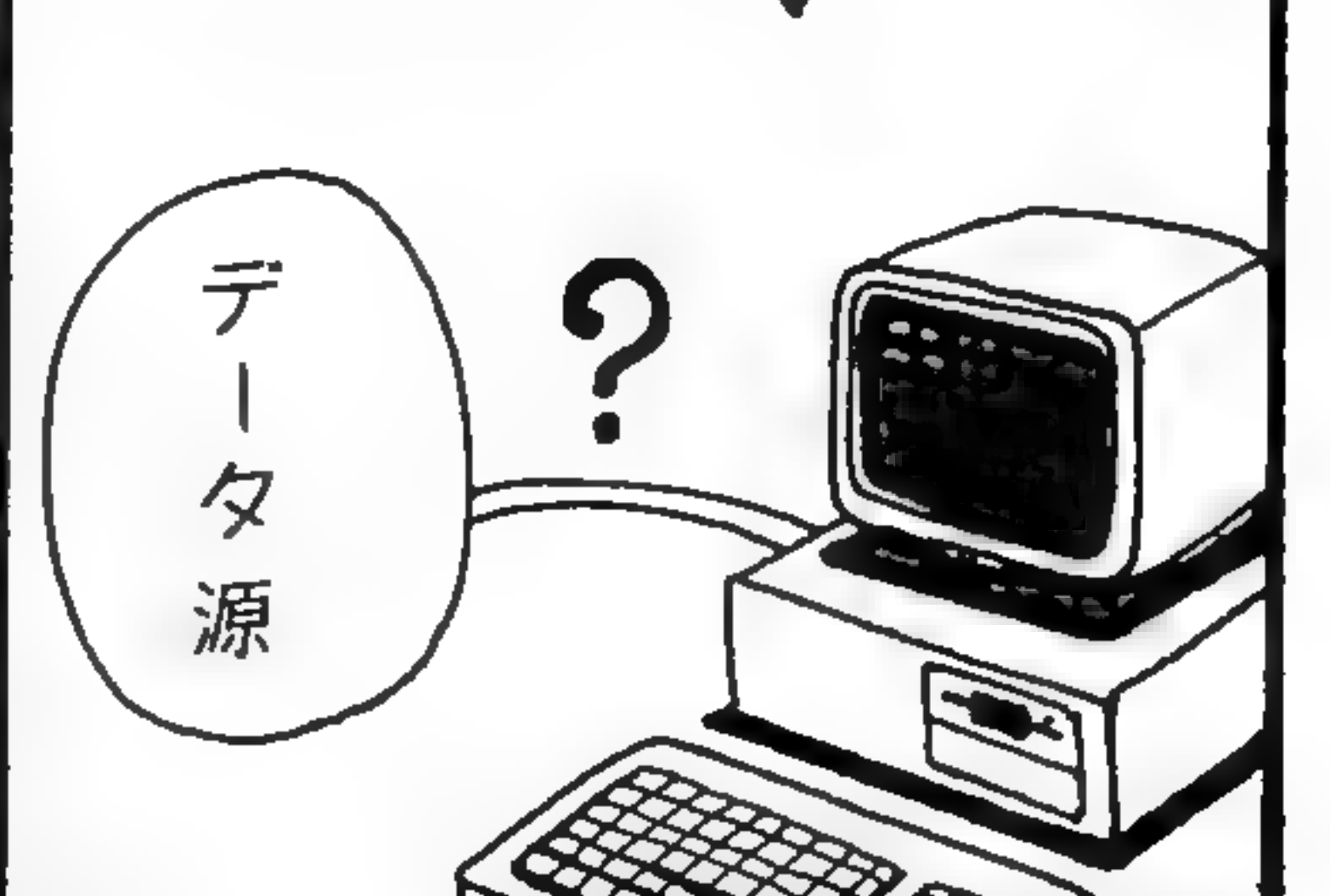
この具体例が、あなたが今持っているニーズやこれからのOA革命対応へのヒントになればと願っています。



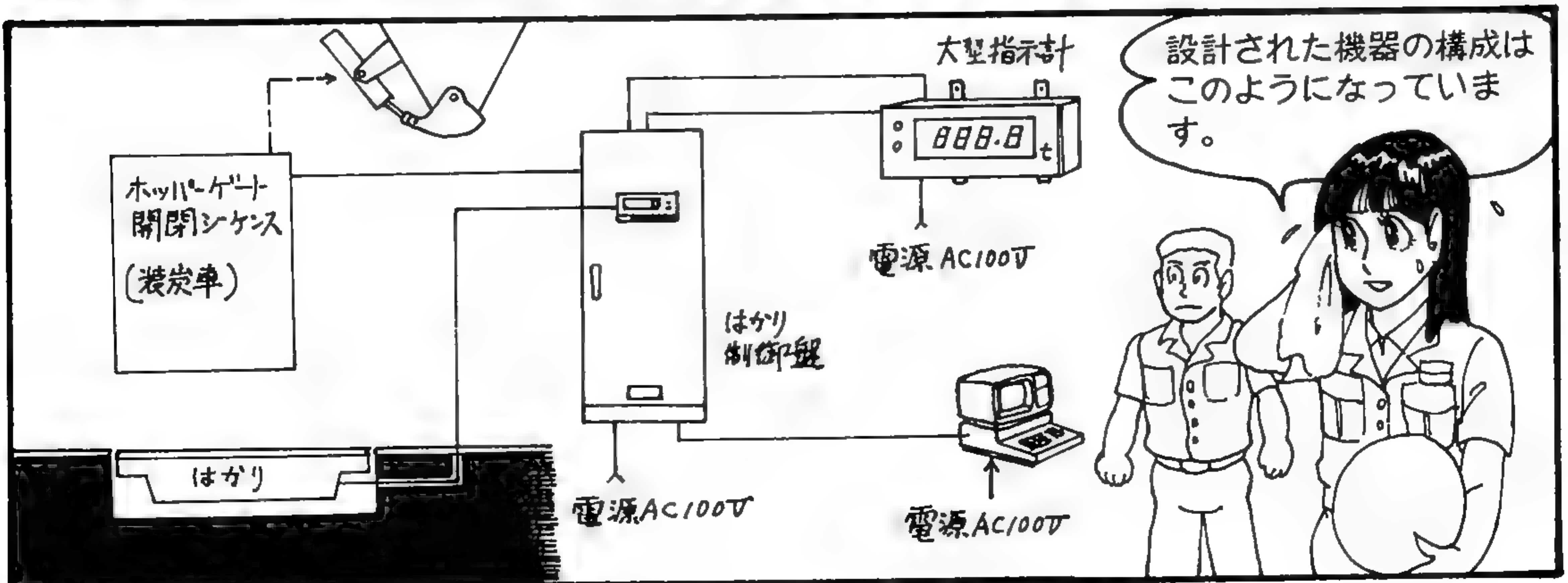
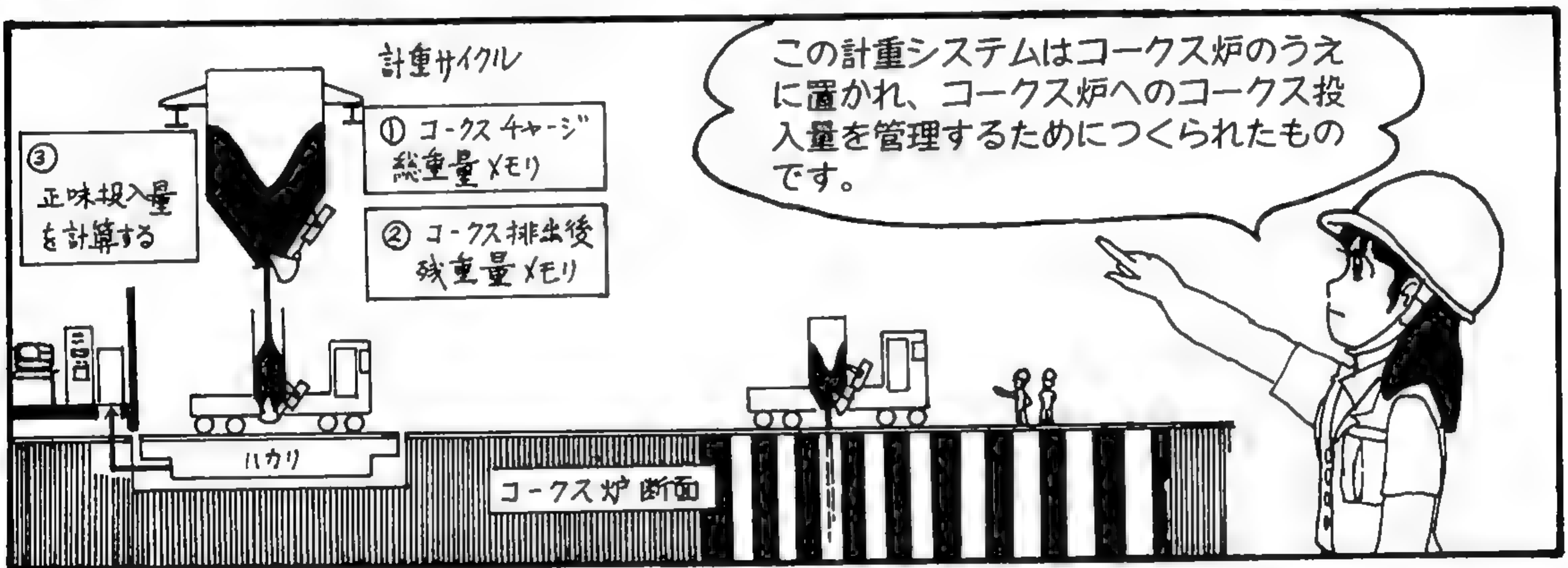
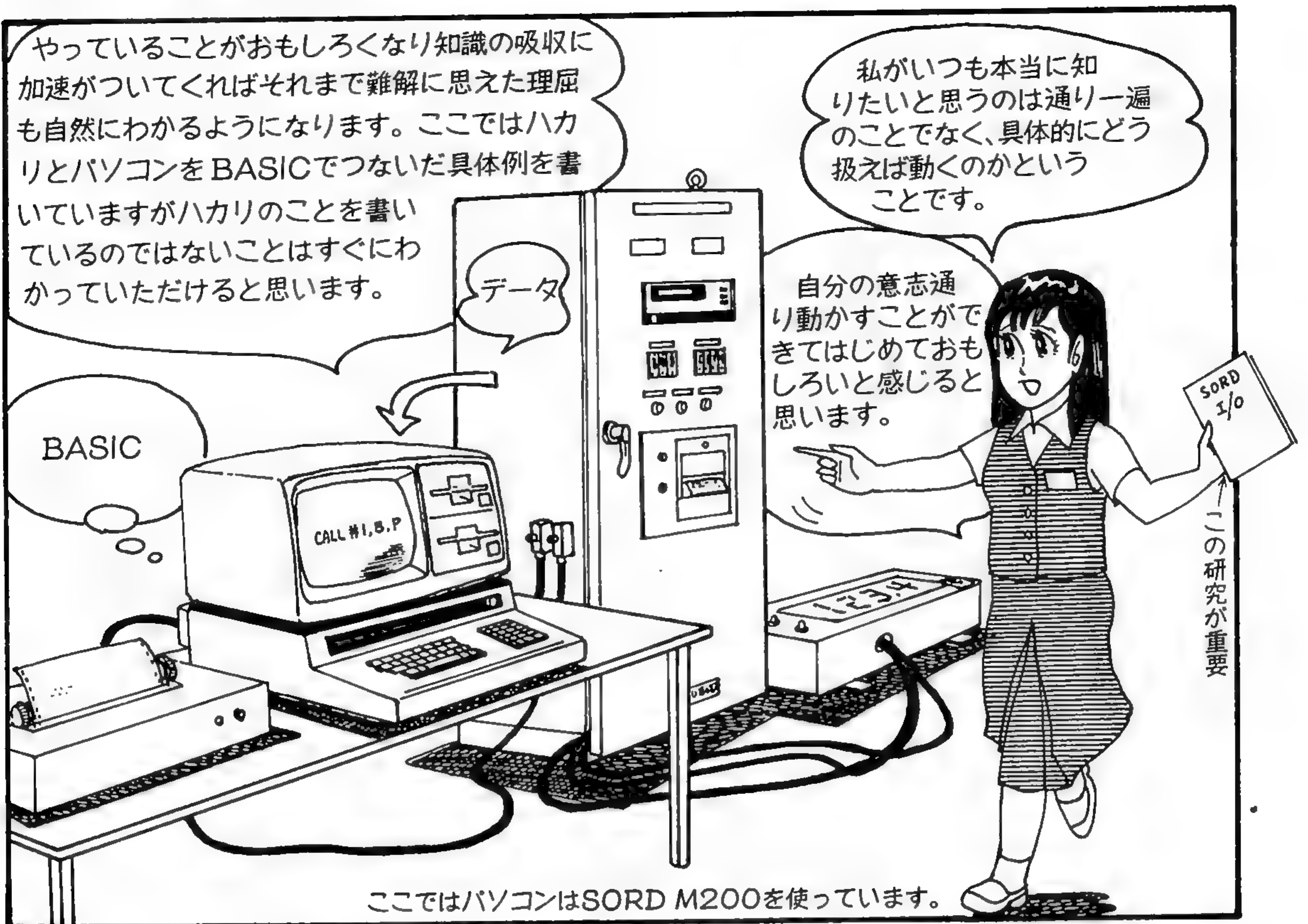
工場の計量器(ハカリ)とパソコンをつないだ例から、なるほどこんな使い方もあるのだということを知っていただきたいと思っています。



コンピュータとデータ源をダイレクトにつなげればOAの効果は増加するはずです。










台車はハカリを出て決められた炉のNo.の穴にコークスを払い出します。

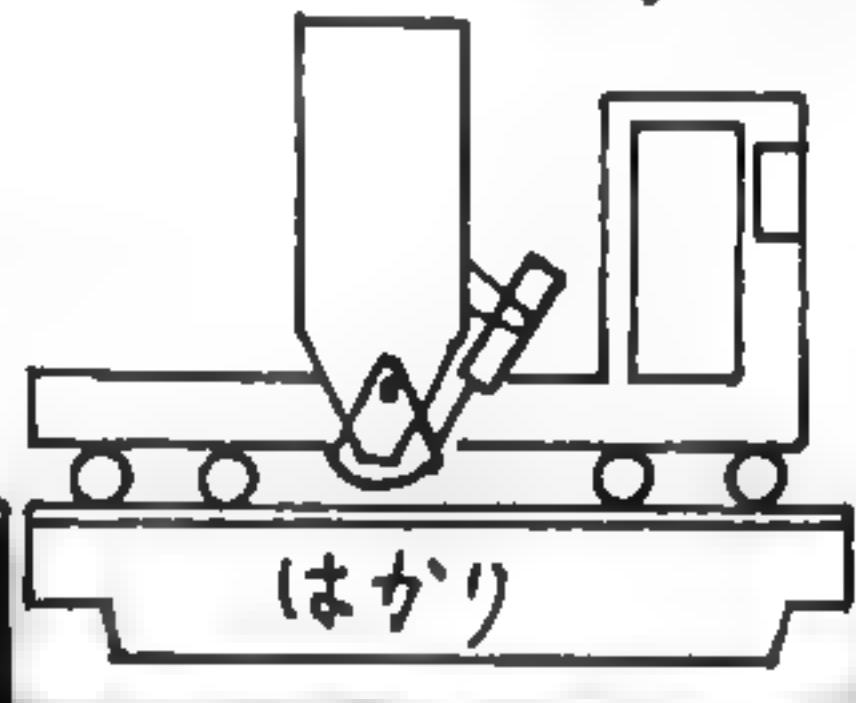


コークスをチャージし、定量まで入れます。そして安定した時のデータを取り込んでこれを総重量 $W_t$ とします。



123.6

ホッパーの下に装炭車が入り、  
コークス投入のボタンを押しま  
す。ホッパーのゲートが開きます。




はかり

85.4

この時が普通のパソコンへの  
データ発信へのタイミングと  
なります。

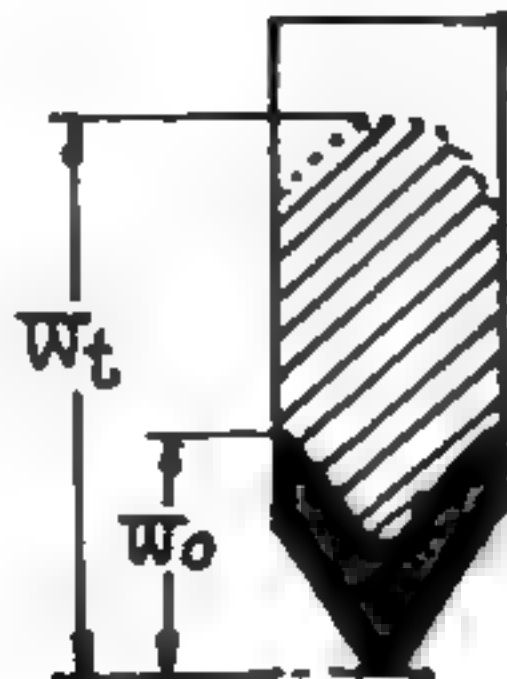
① 炉番 3桁  
② 総重量 4桁  
③ 残重量 4桁  
④ 正味重量 4桁

②と③から④は計算でき  
るが照合用として発信する。

A black and white illustration of a woman with short dark hair, wearing a dark vest over a light-colored shirt. She is pointing her right index finger upwards towards the text above her.

この時、炉に払い出したコークスの正味重量が計算できます。(Wn)

$$W_n = W_t - W_o$$



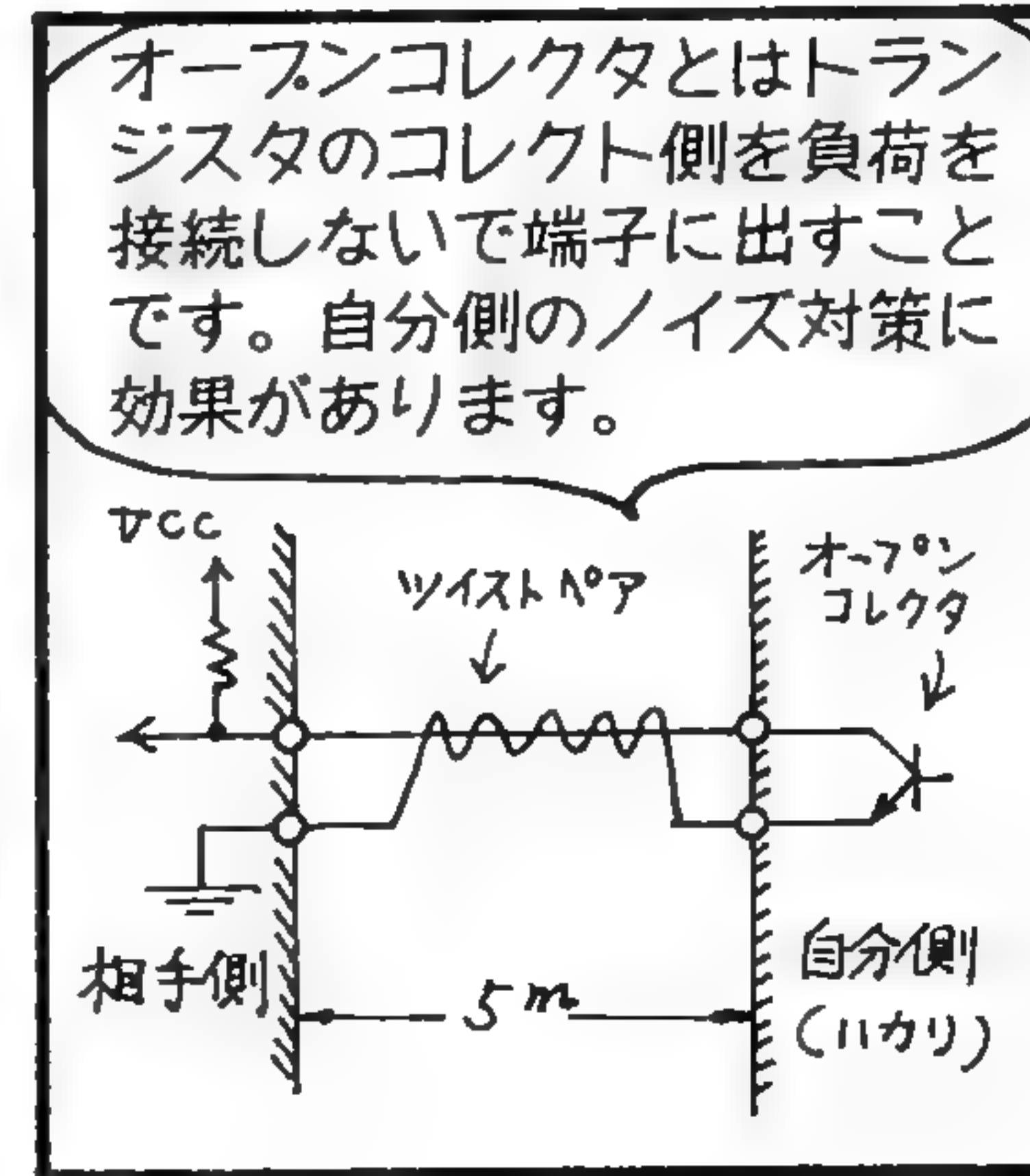
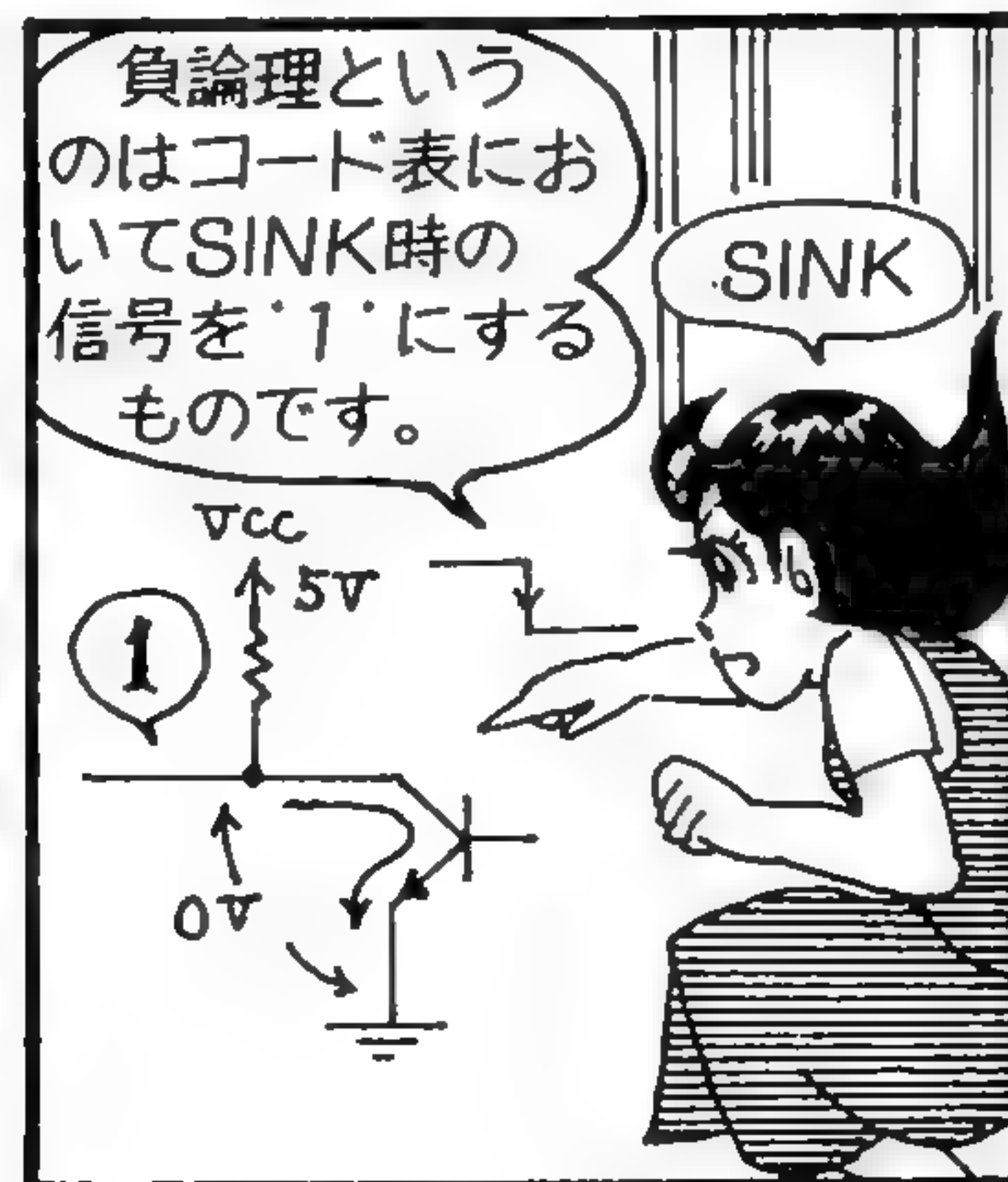
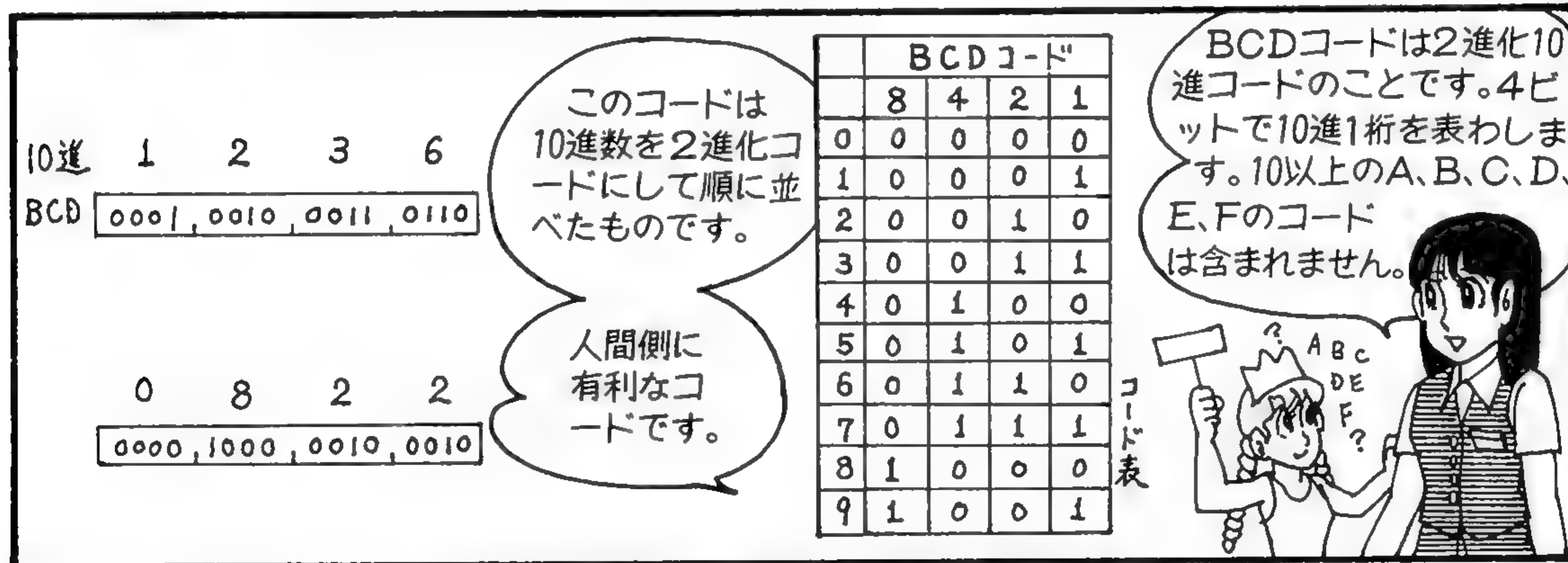
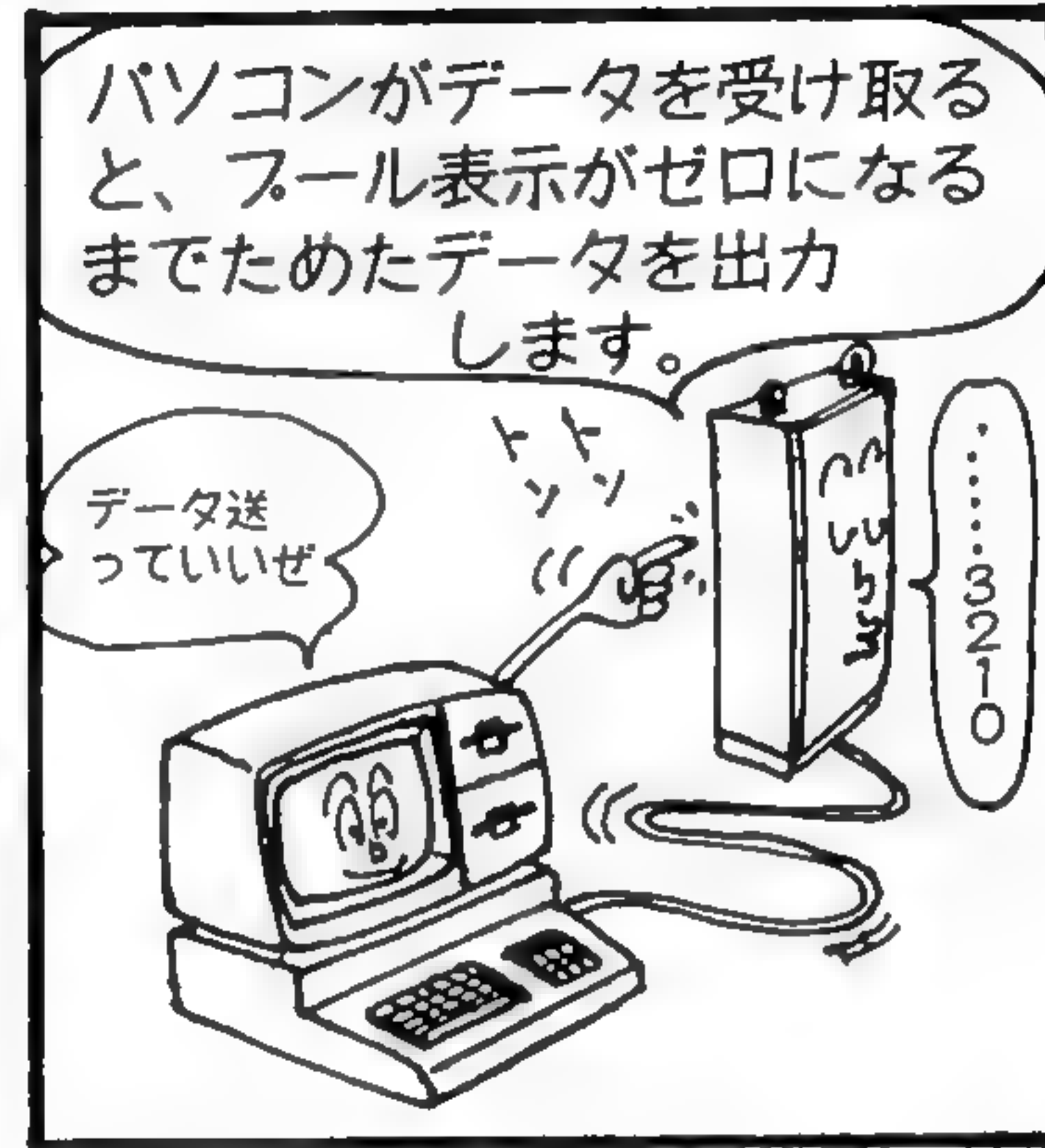
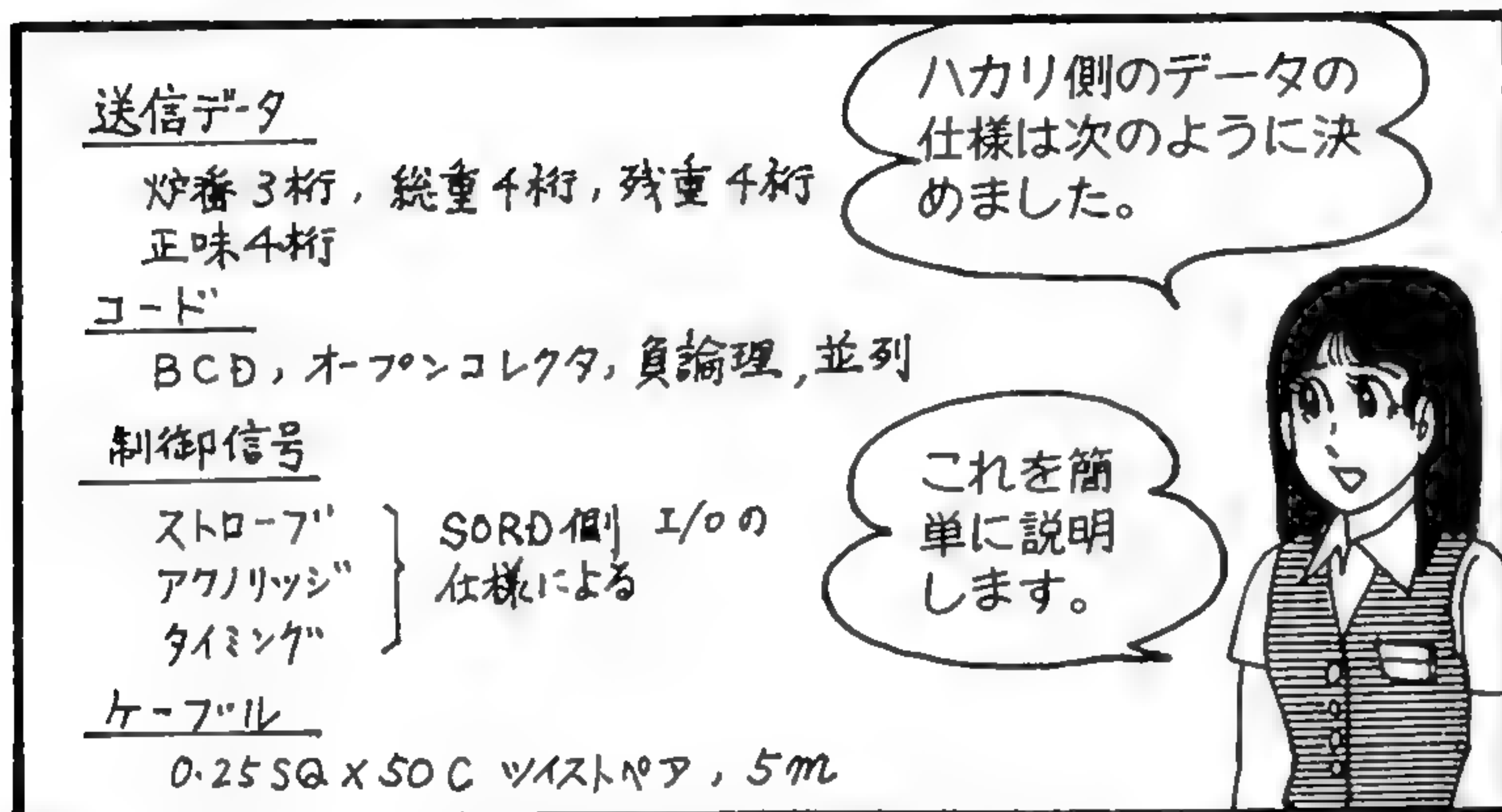
The diagram shows a vertical pile of coke. The total height of the pile is indicated by a vertical dimension line on the left, labeled  $W_t$ . The height of the bottom, conical portion of the pile is indicated by a smaller vertical dimension line, labeled  $W_o$ . The top portion of the pile is filled with diagonal hatching lines.

そして装炭車は再びハカリに戻りそのまま計重します。  
この時のデータを残量 $W_0$ として取り込みます。



B2.2



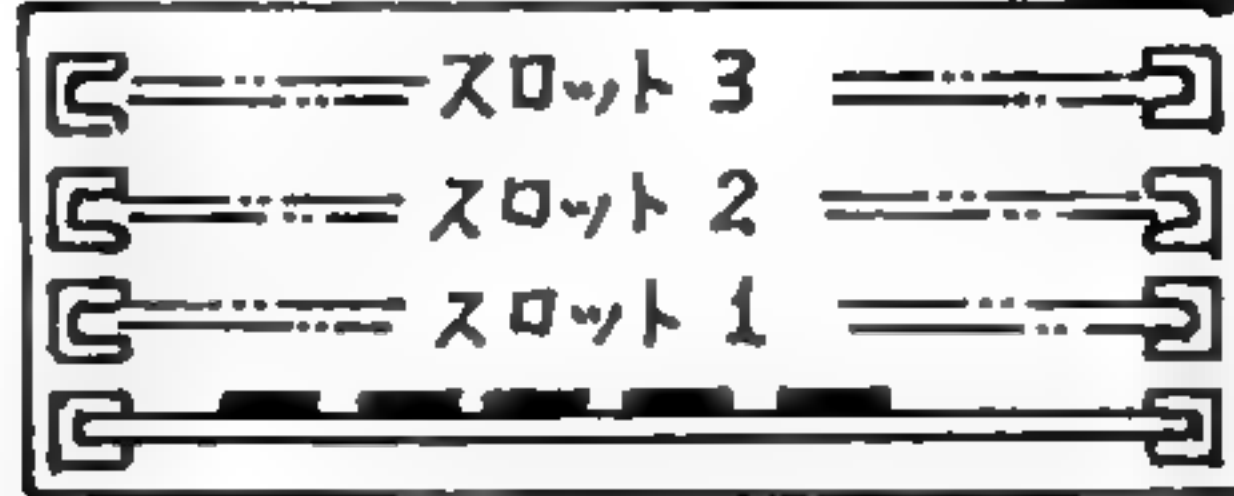




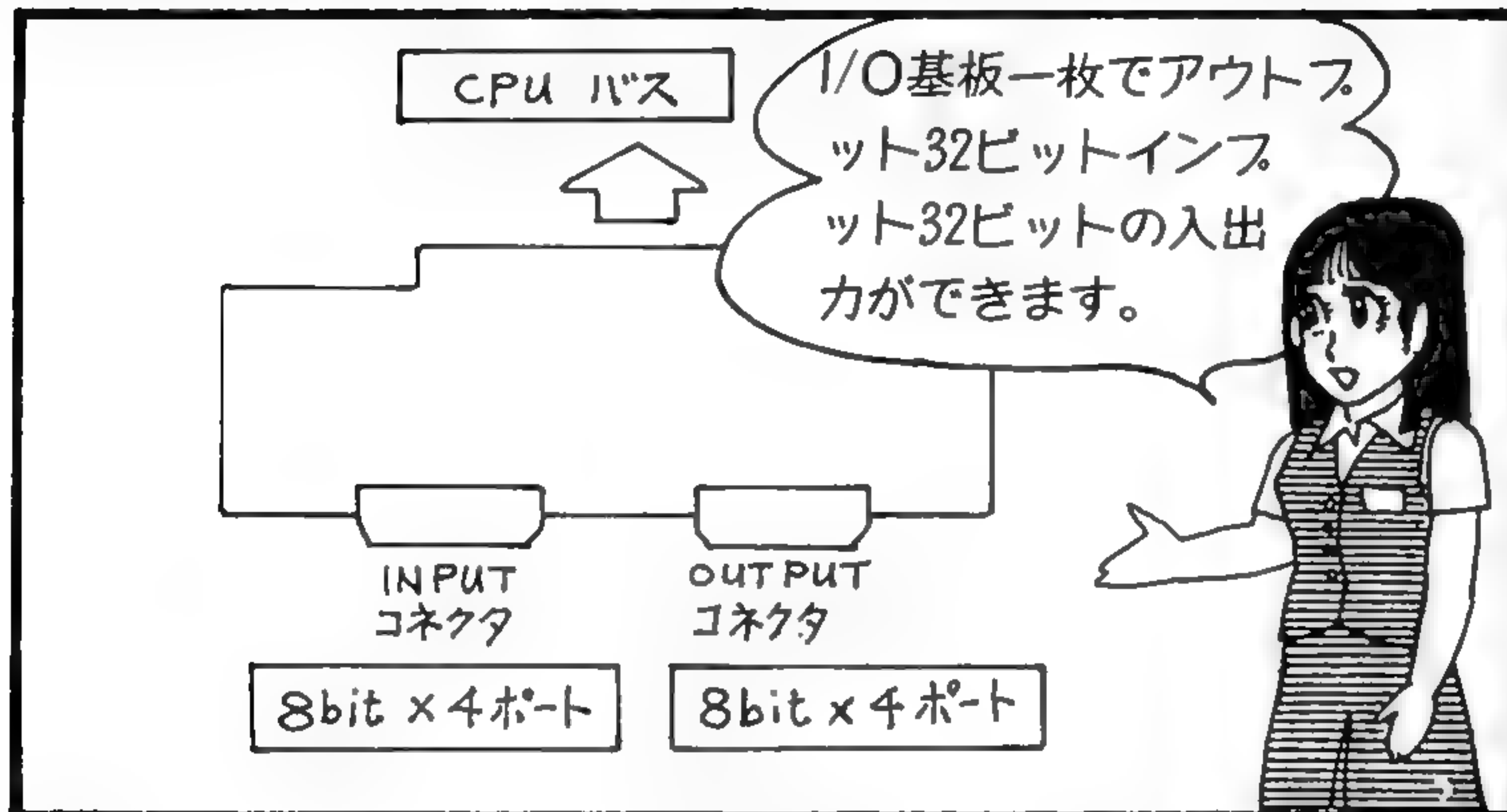
ここにI/O基板を入れ、ハカリ側からのデータを受け取れるようにするのです。



一番下にすでに基板が一枚入っています。そして、その上にさらに3枚分の基板を入れるスペースがあります。



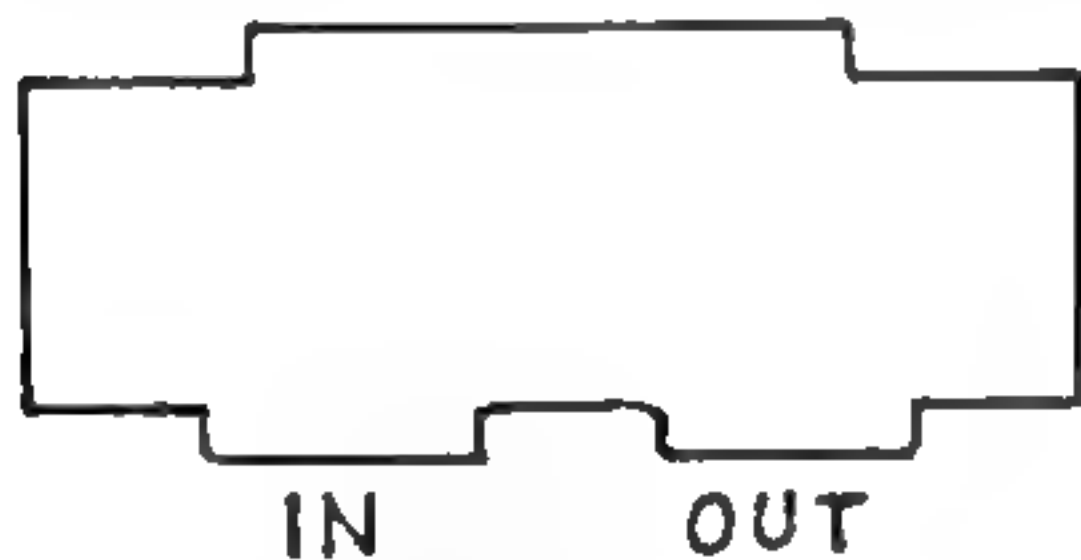
次はパソコン側です。裏側のフタをはずしてみます。



これを使うためには前に紹介したI/Oマニュアルを読まなければなりません。

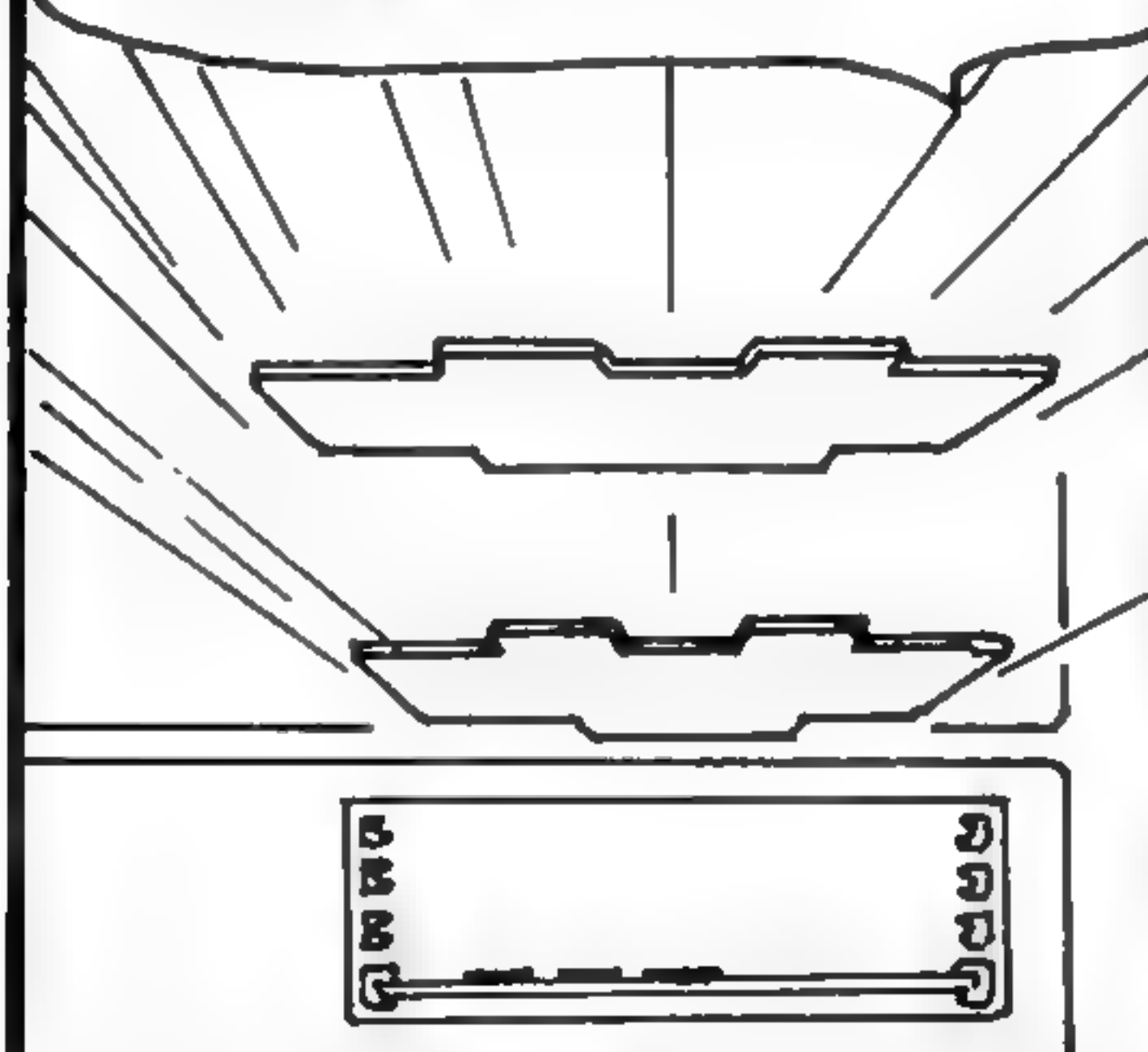
PART: 2  
HC-DIO  
M200用デジタル入出力ユニット  
ハードウェア・ソフトウェア 説明書

データはパソコンにとってはすべてインプットになるため、アウトプット側は使いません。



こっちだけを考える

ハード的にはこのI/O基板が2枚必要になります。



ハカリ側からのデータは全部で60ビットになります。BCDコードで15桁分です。これを受け取るためには、

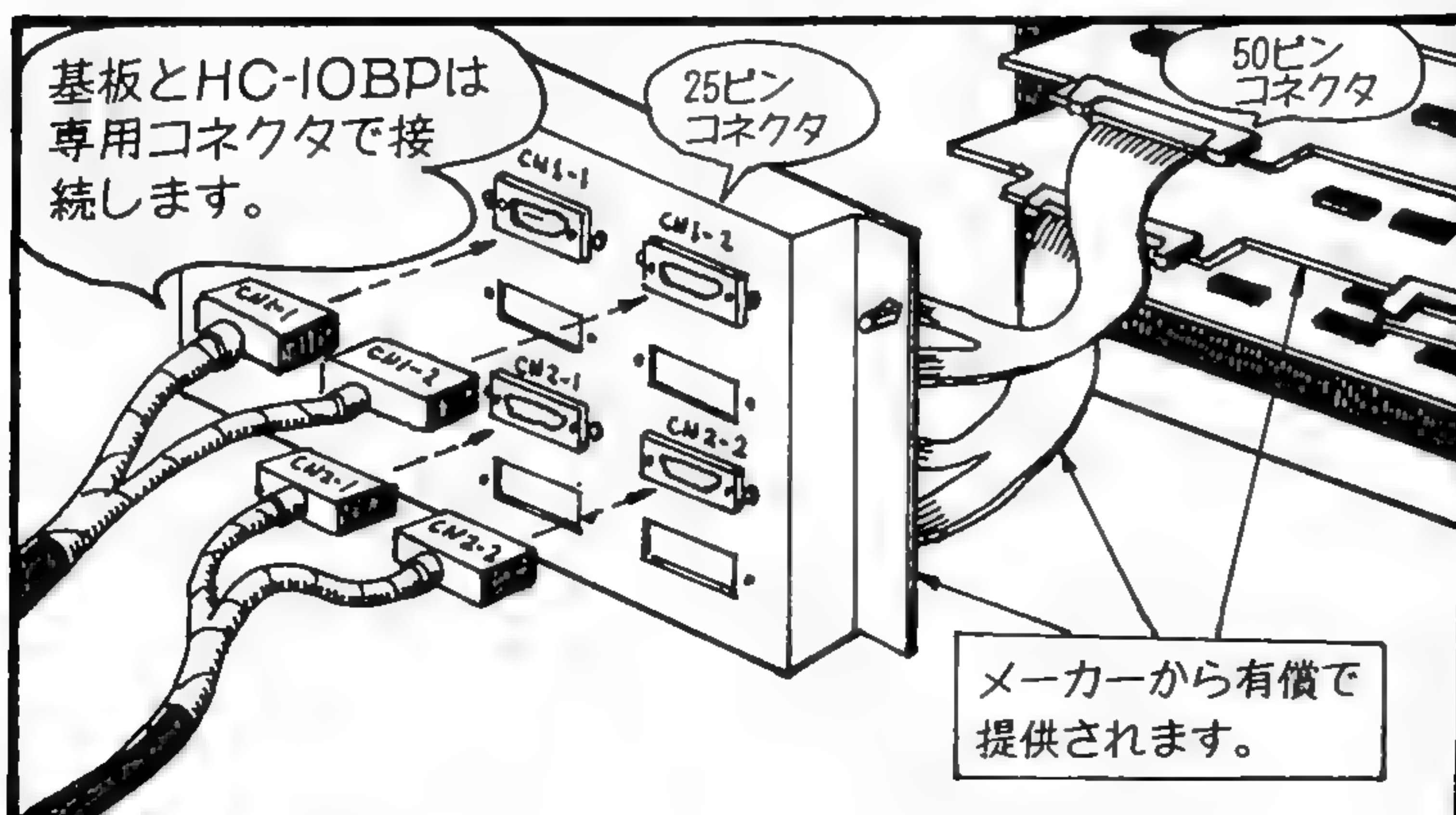
炉番 3桁  
総重 4桁  
残重 4桁  
正味 4桁

計 15桁

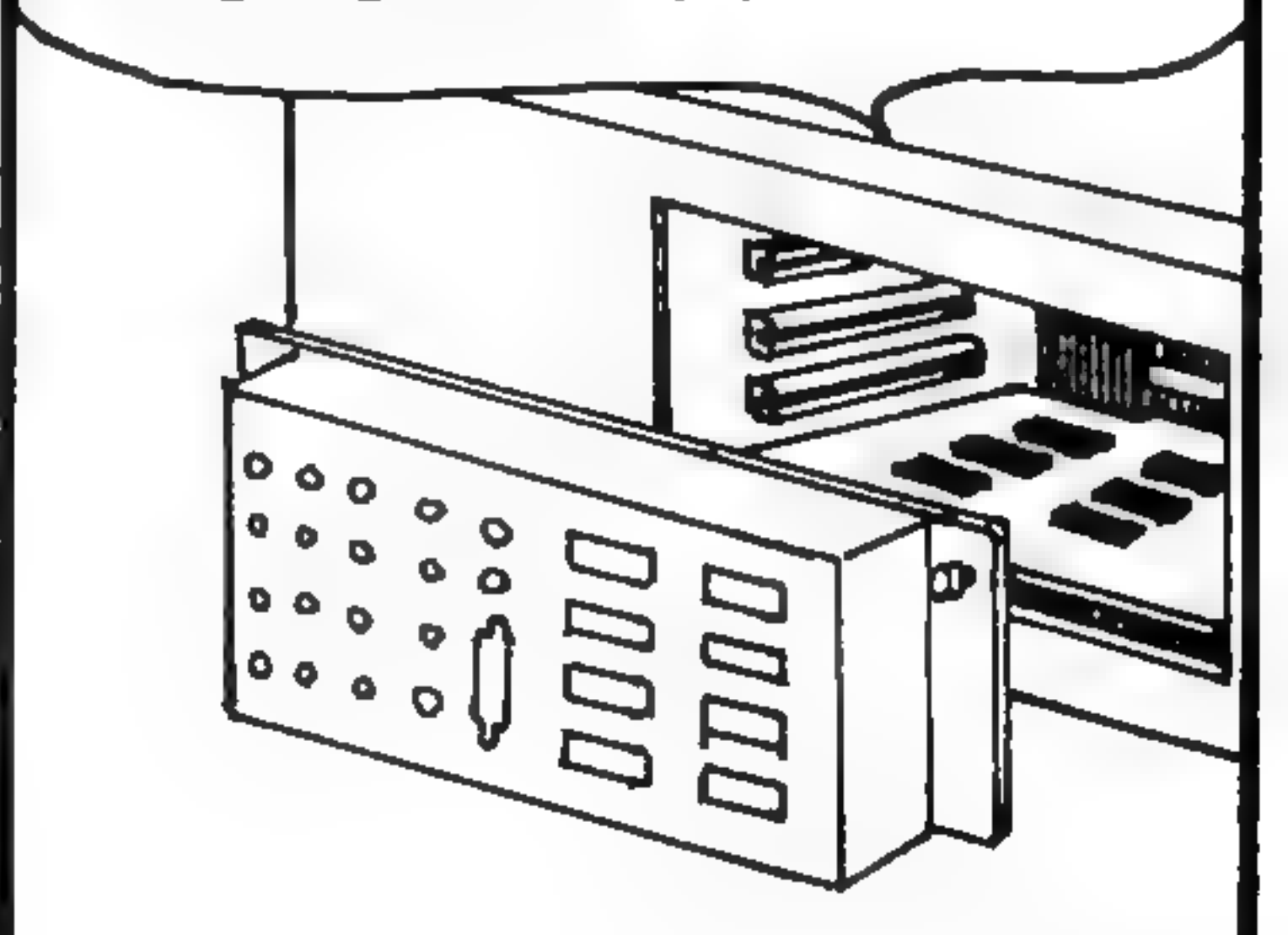
15桁 x 4bit  
= 60bit



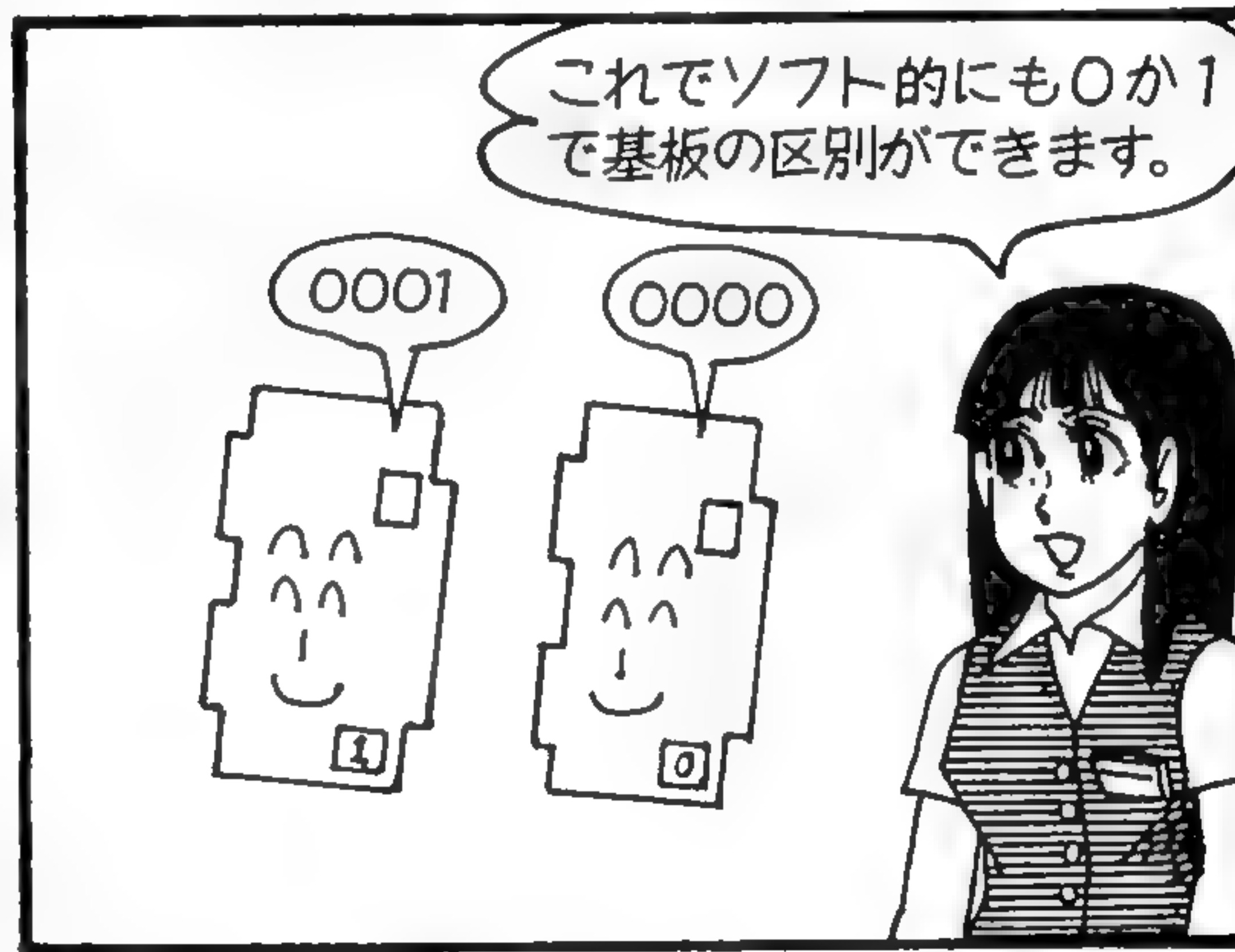
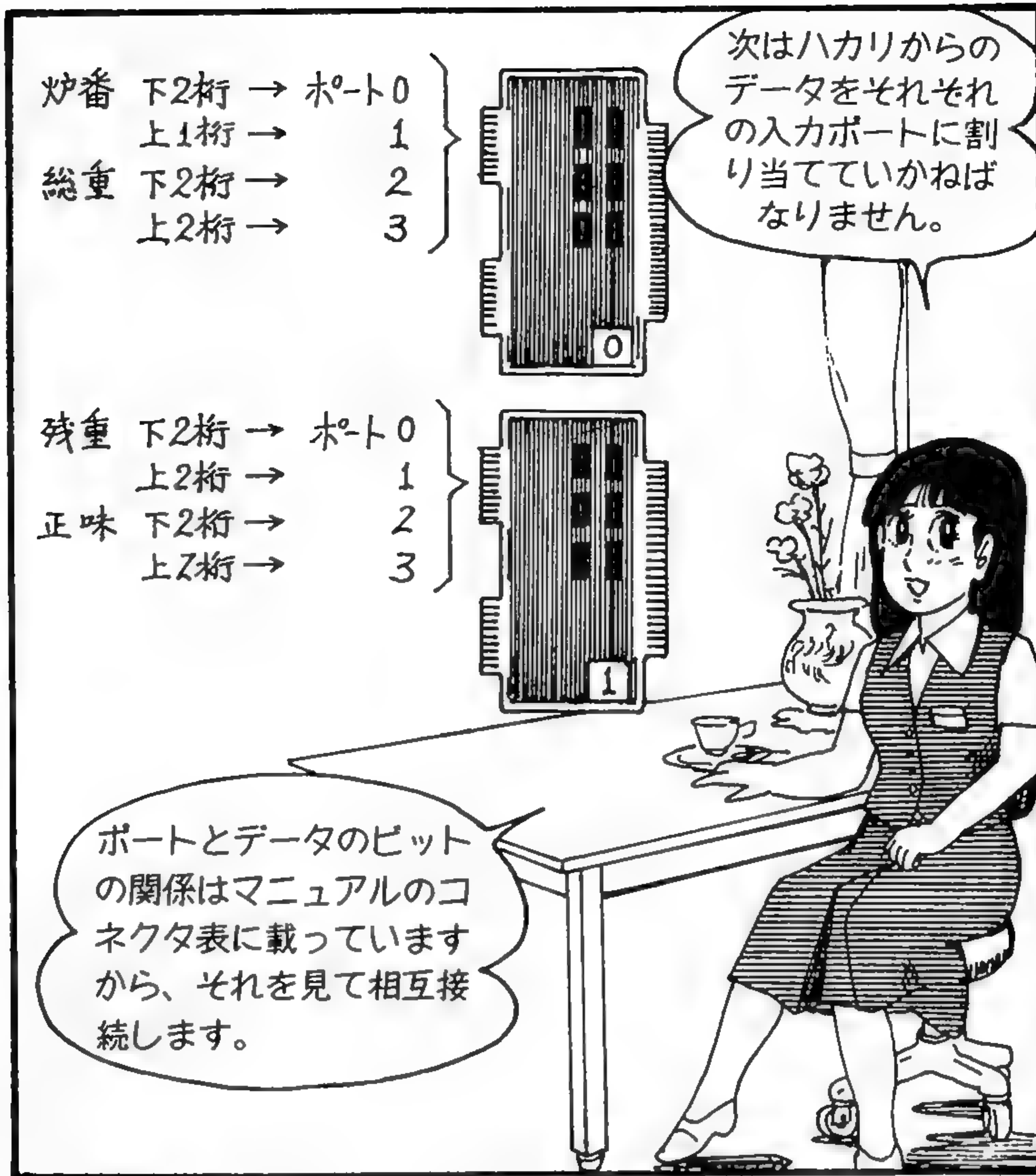
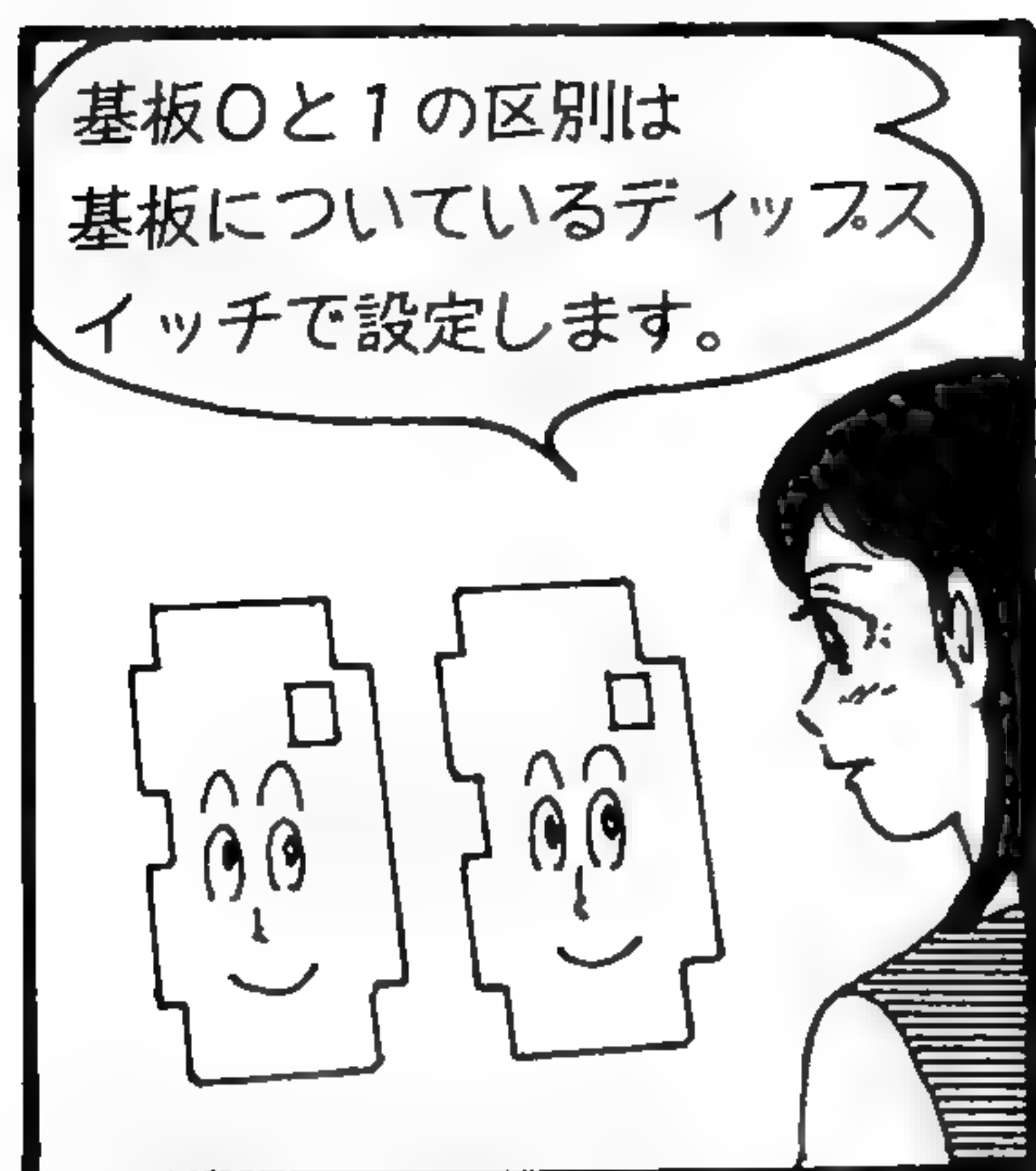
基板とHC-IOBPは専用コネクタで接続します。



データ取出しのためにコネクタが必要です。そのコネクタ取付パネルとして用意されたのがこのHC-IOBPです。

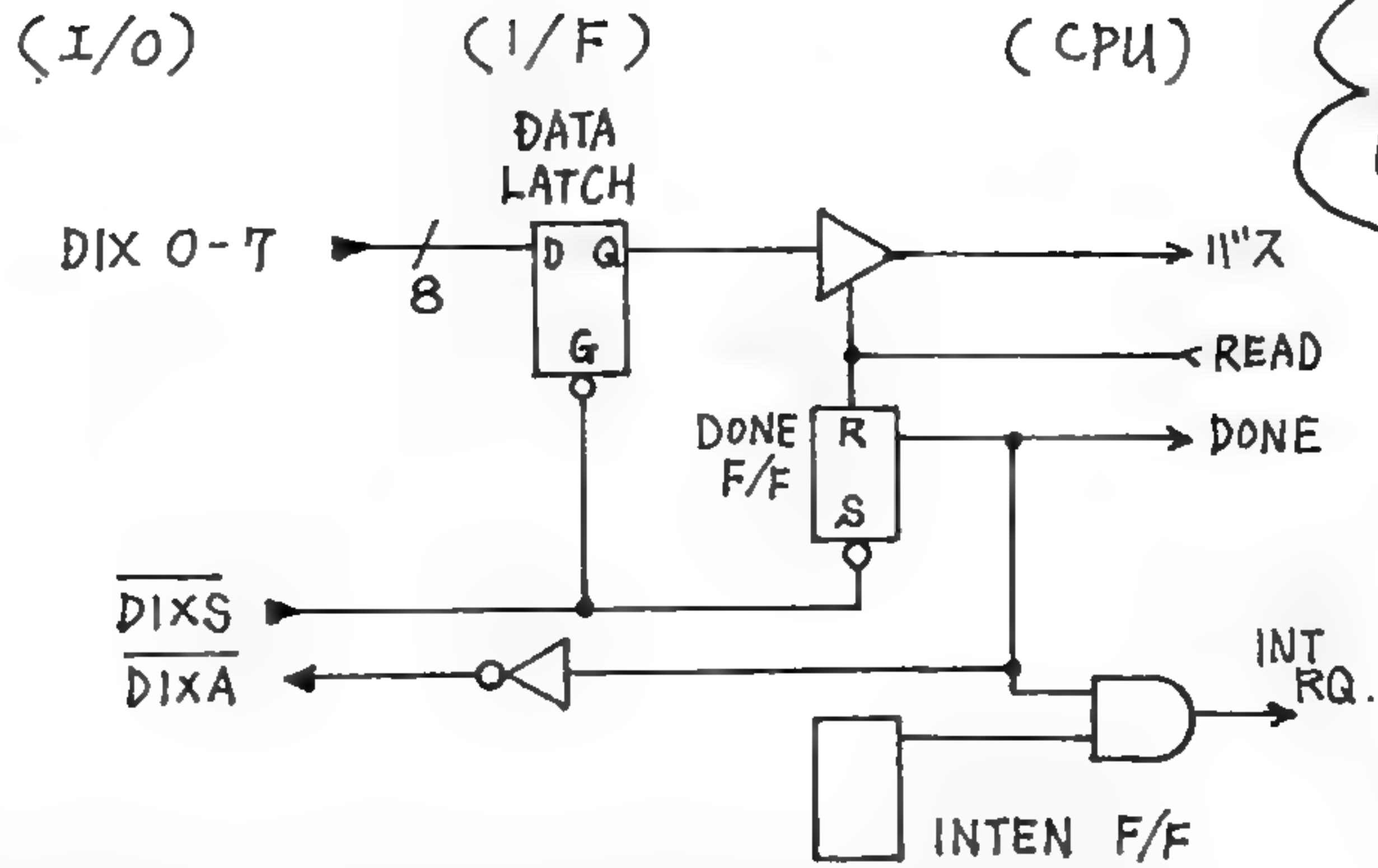




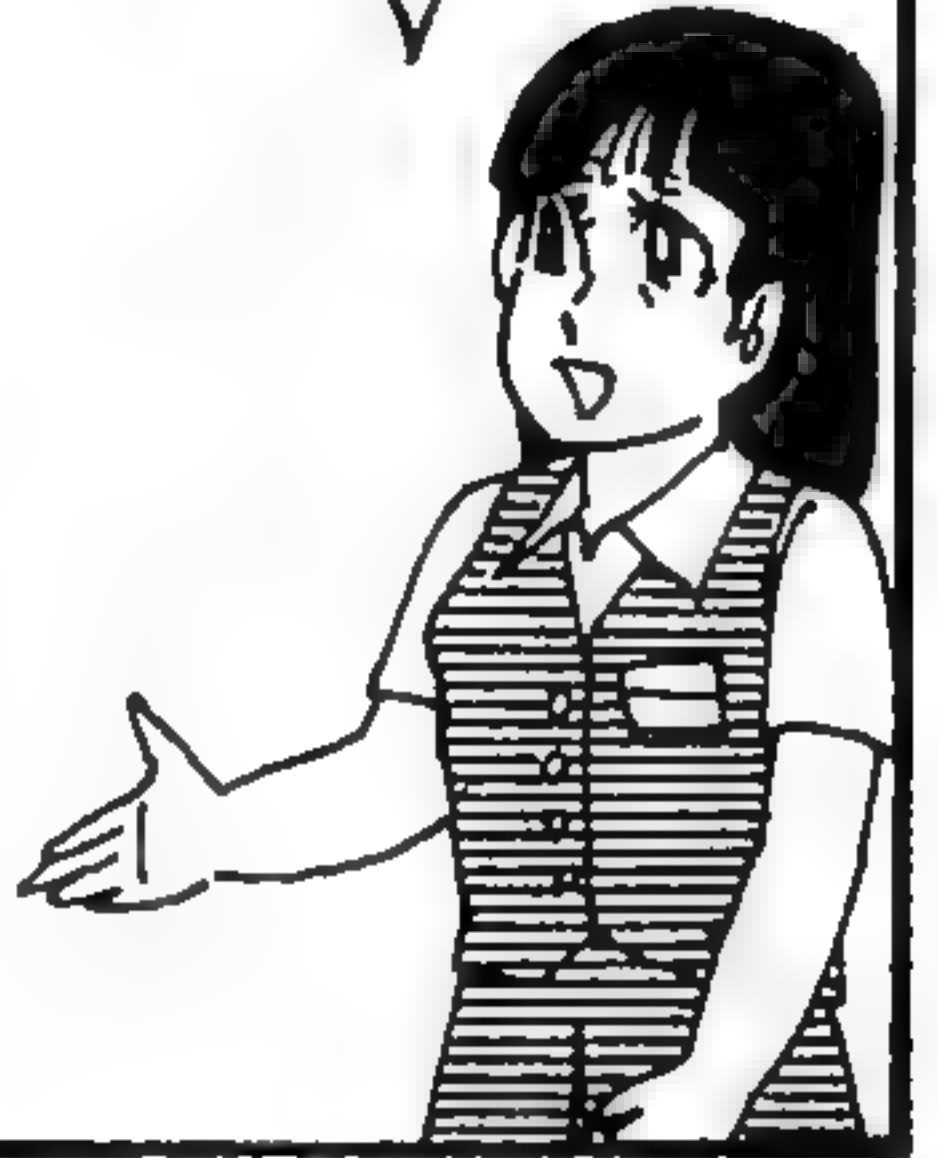




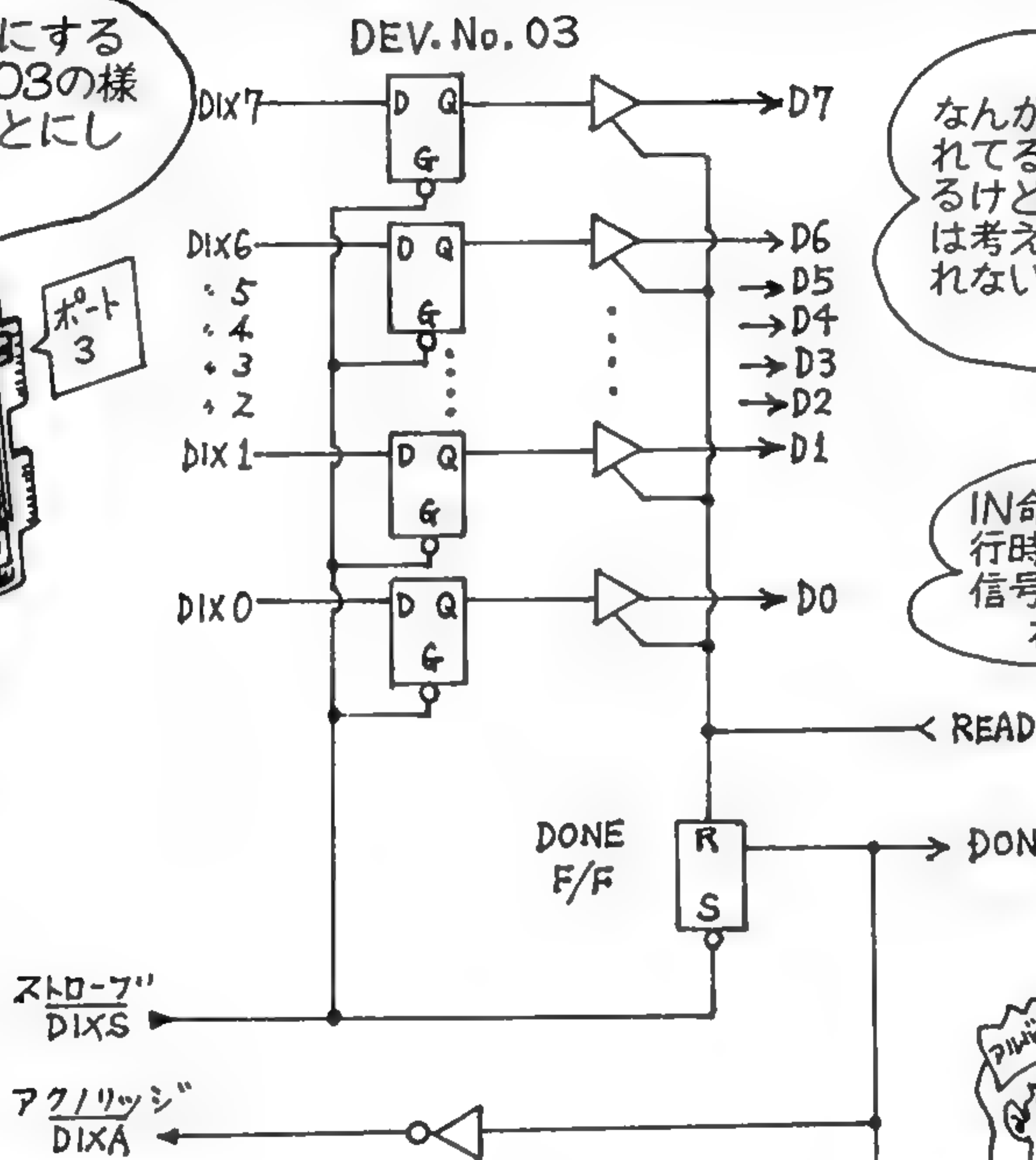
ハードのことが全然  
わからない人には  
この図の解説はむずか  
しいと思います。



マニュアルの説明  
図をそのまま転記  
しました。



具体的な内容にする  
ためにDEV.No.03の様  
子を見てみることにし  
ます。



なんか無視さ  
れてる感じがす  
るけど僕抜きで  
は考えら  
れない。

そこでこのマン  
ガ風書き直し  
てみました。

僕A  
レジスタ

IN命令実  
行時のINP  
信号と考えら  
れる

4番目のビットに'1'書  
き込み

7 6 5 4 3 2 1 0

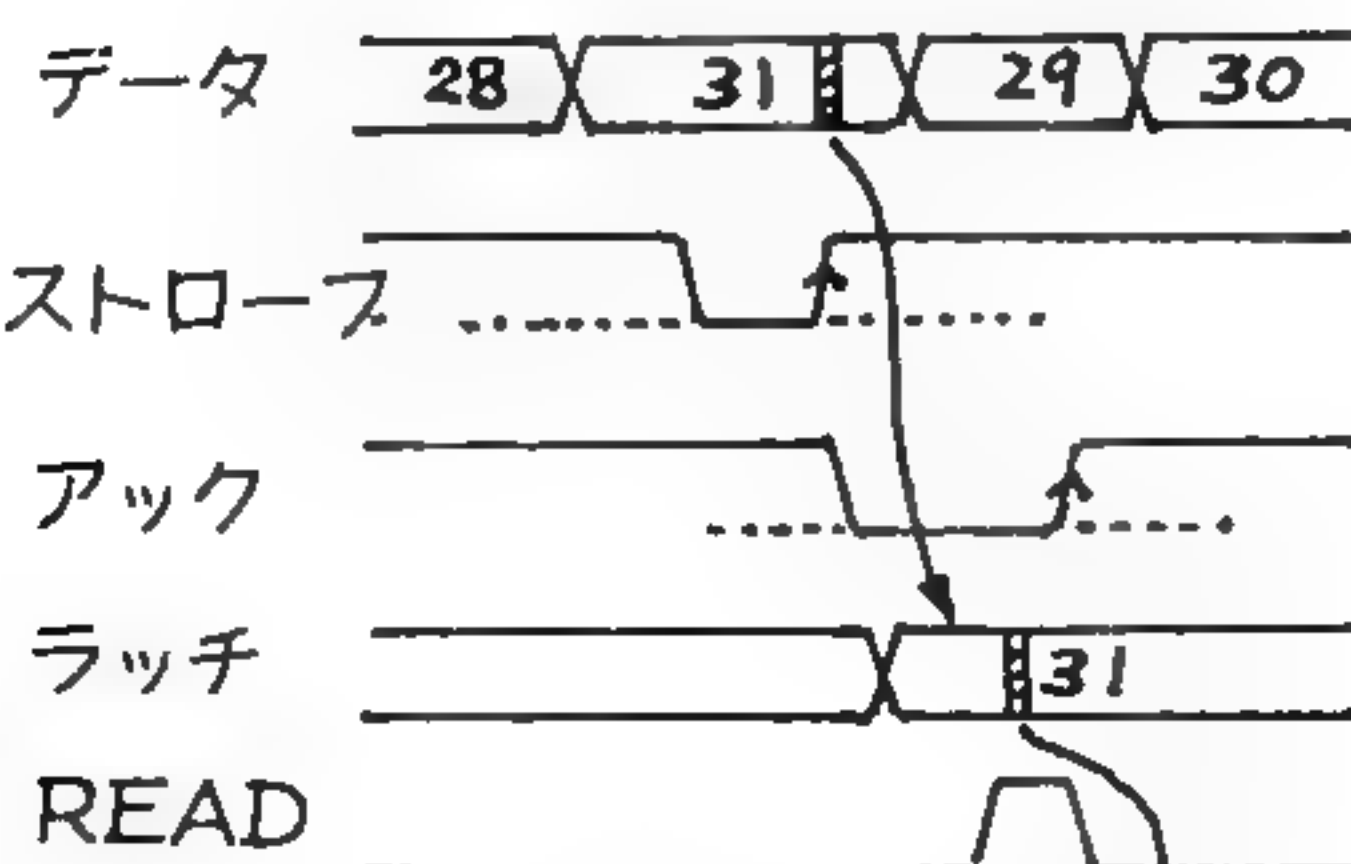
↓ Aレジ  
スタへ

DEV.No.

04

ソフト的には、  
CALL文実行  
時このレジスタ  
を参照している。

データのチェンジ  
を示します。



CPUのAレ  
ジスタには31  
が入る

I/O命令下の  
レジスタとして  
DEV.No.0Bで  
書き込みできま  
す。

7 6 5 4 3 2 1 0

INTEN  
F/F

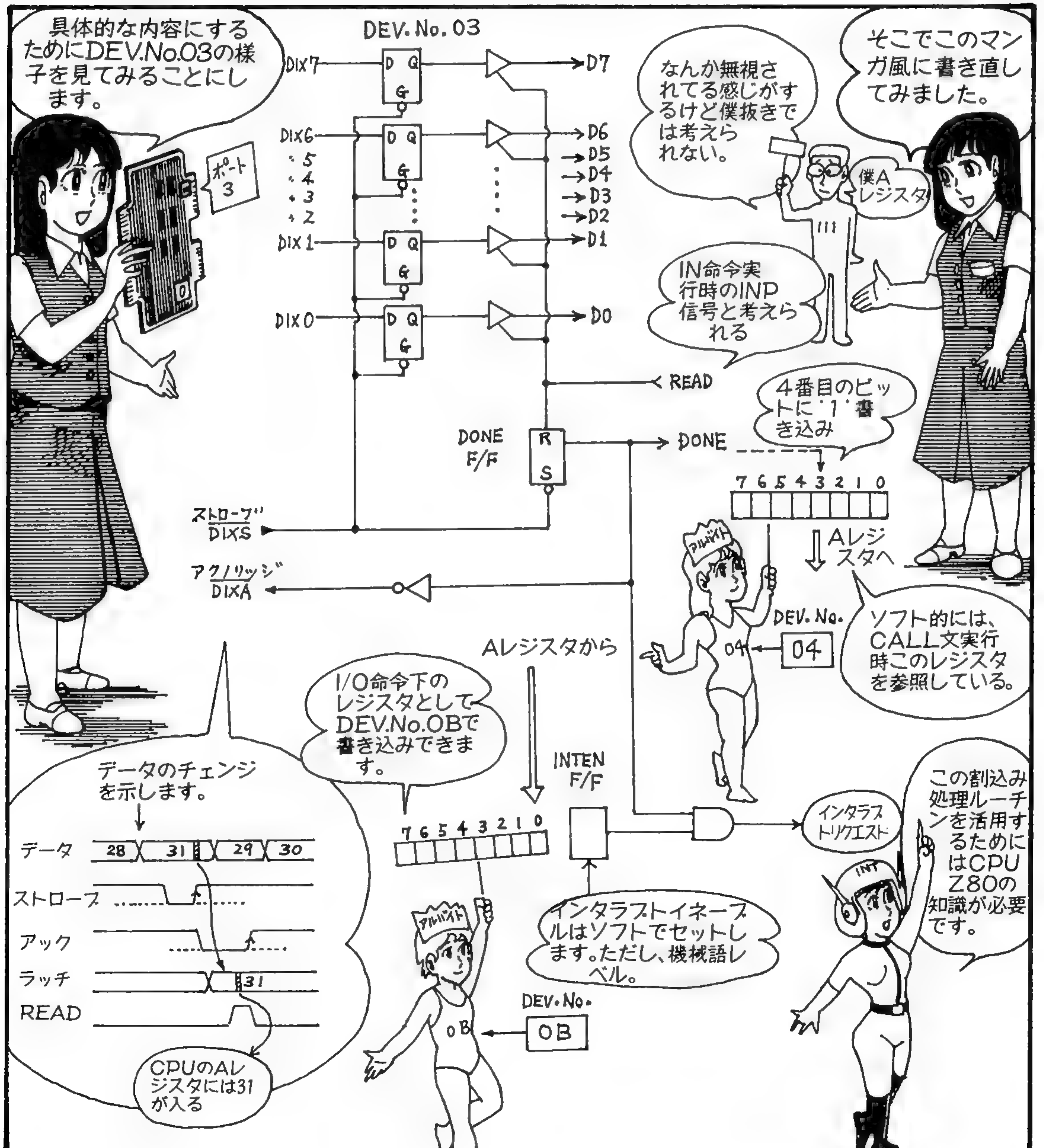
インタラフトイネー  
ブルはソフトでセッ  
トします。ただし、機  
械語レ  
ベル。

DEV.No.

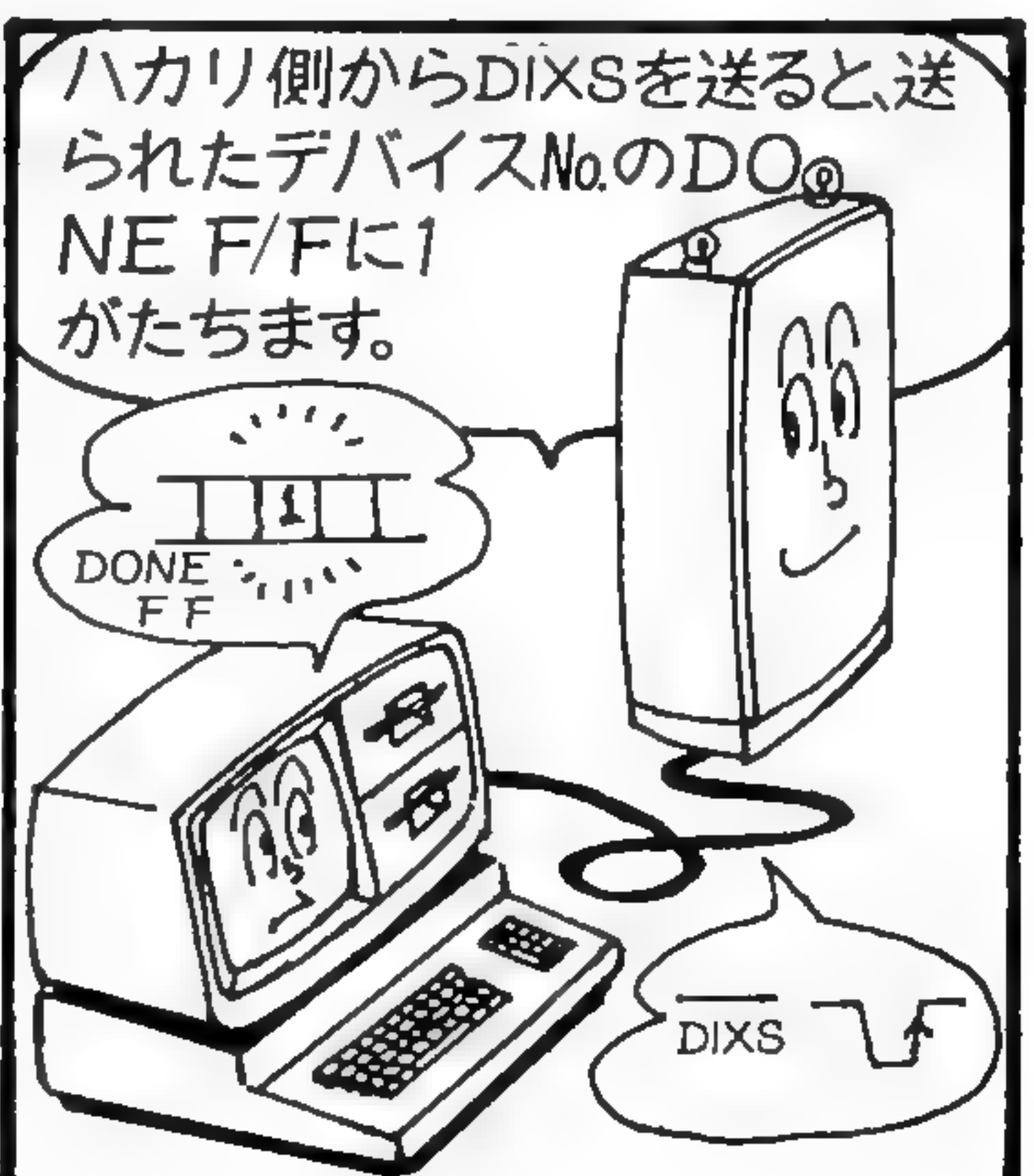
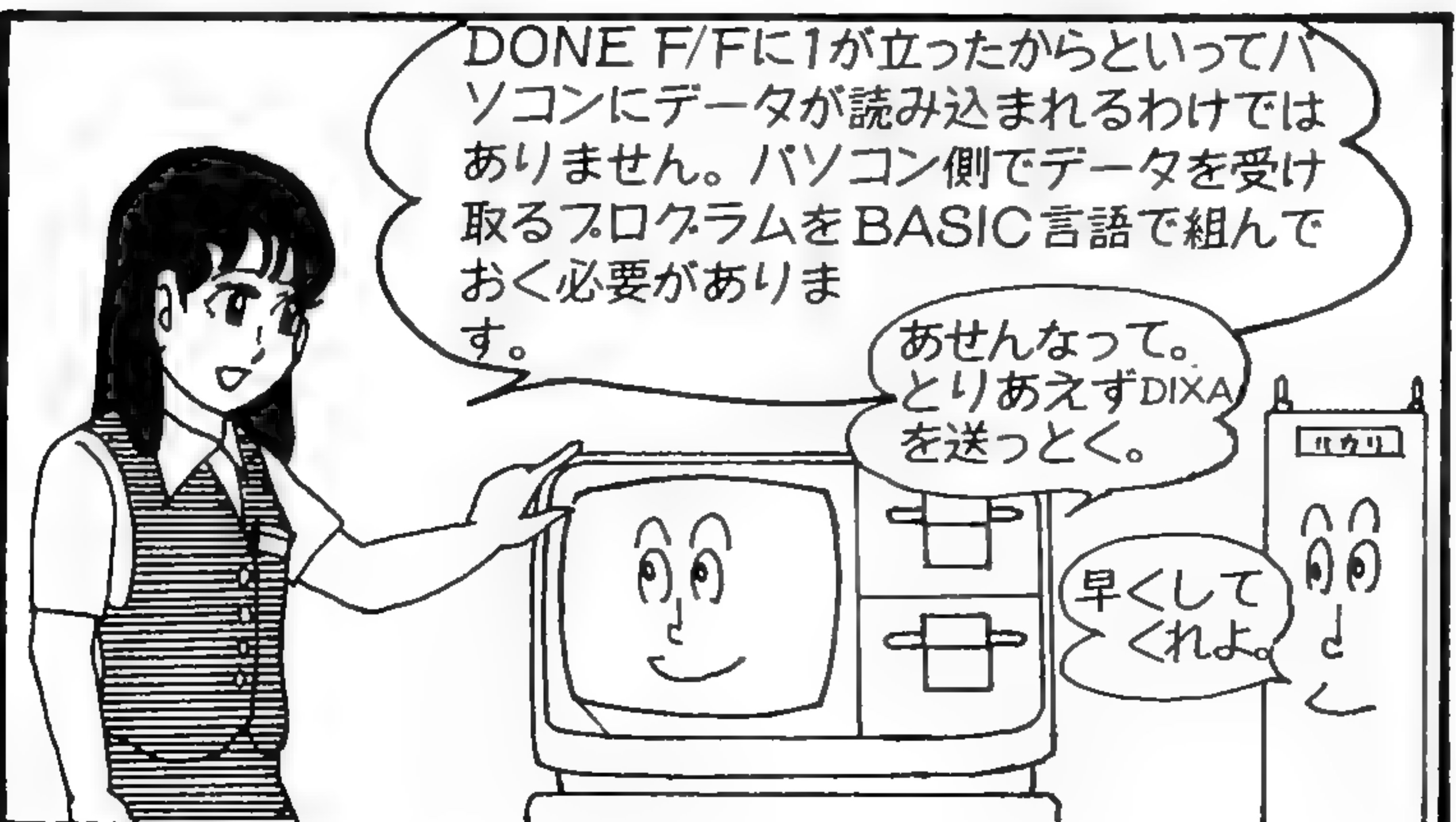
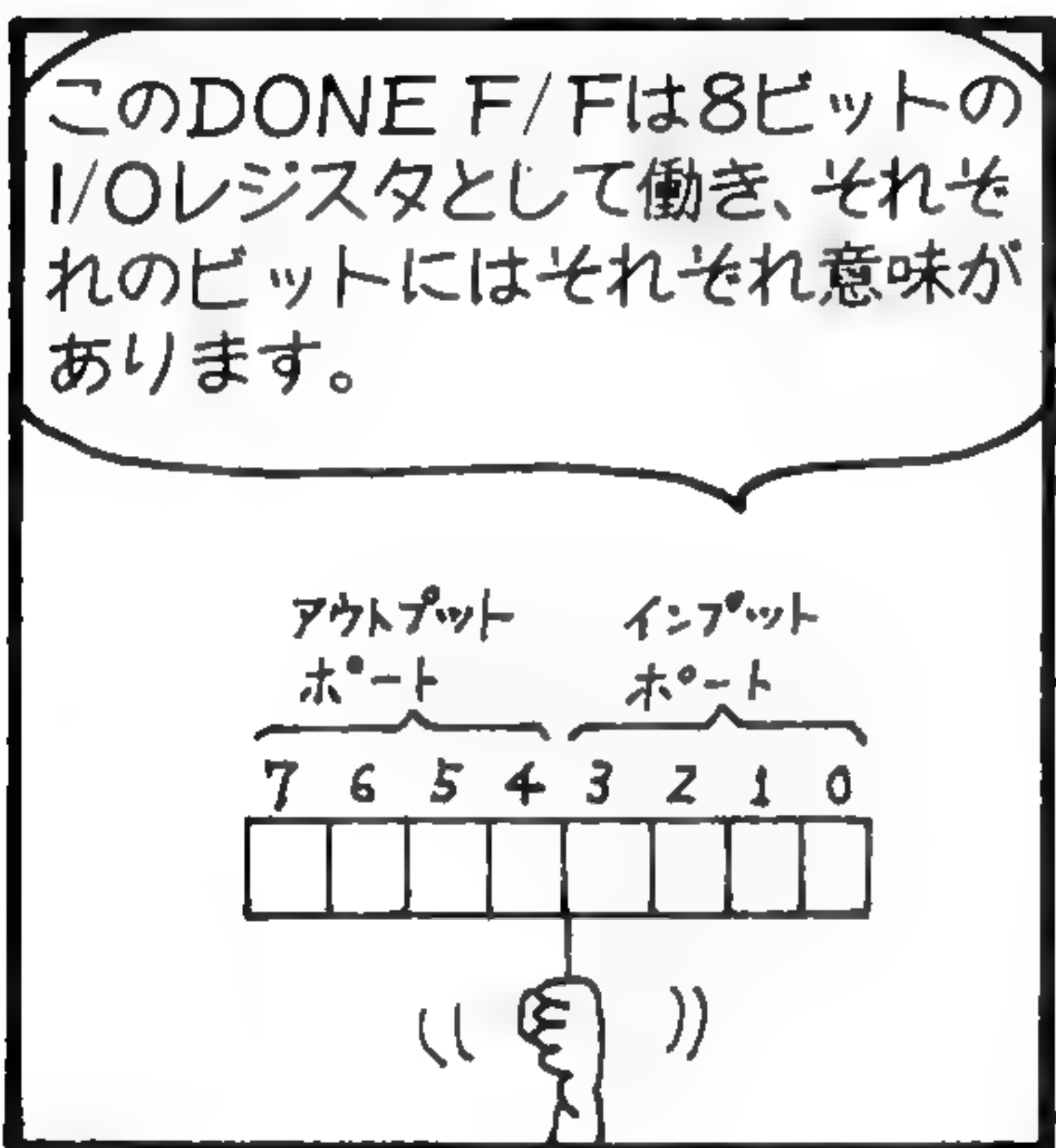
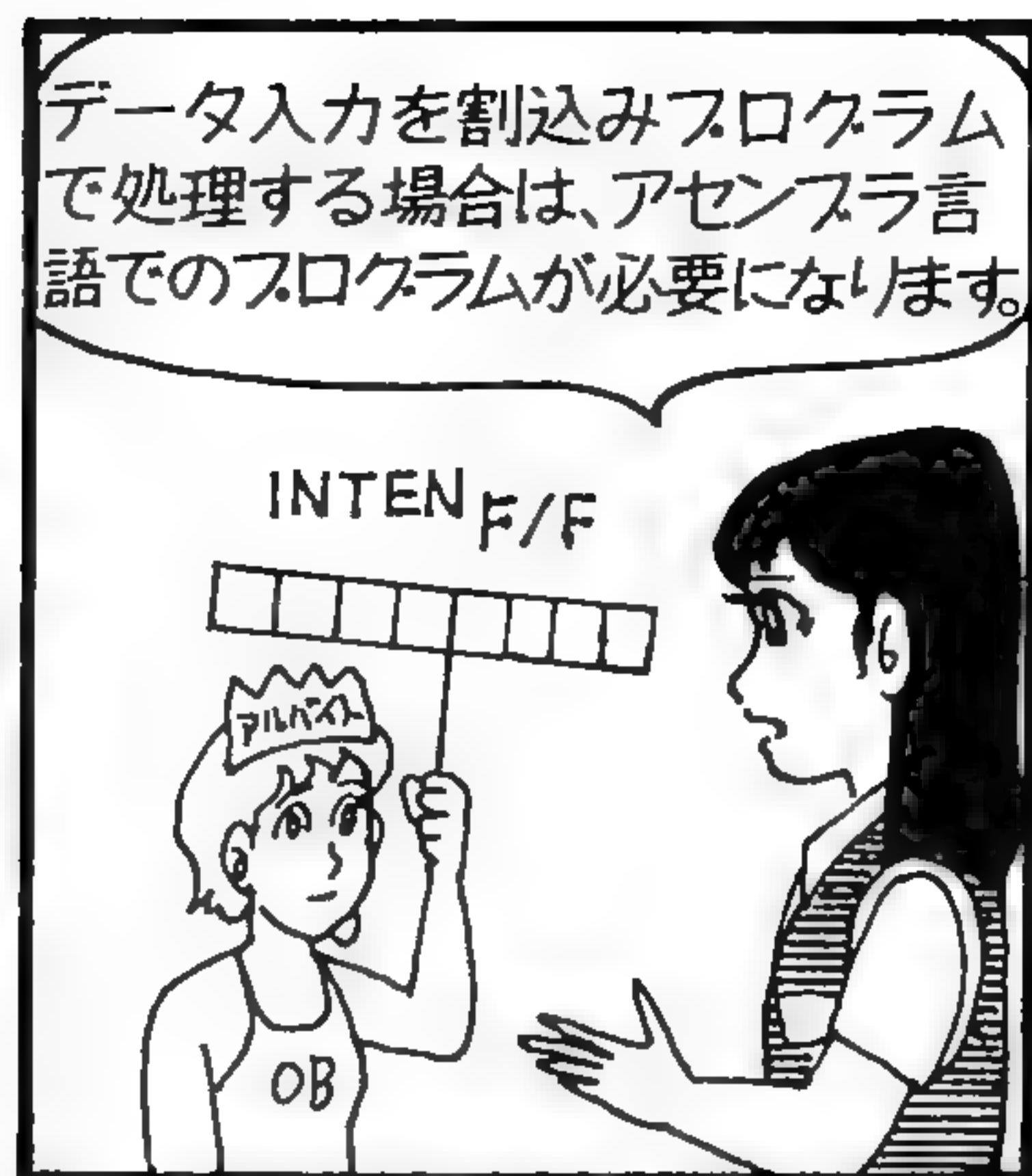
0B

インタラ  
ストイレ  
スト

この割り  
込み  
処理ルー  
チンを活  
用するた  
めには  
CPU  
Z80の  
知識が必  
要です。









これだけではなんのことがわかりませんので、ハカリのデータインプットを例にご説明します。

これがその文の書き方です。

CALL #n, argument1, argument 2

ではいよいよプログラムです。BASIC言語で外部からデータをインプットするんですよ。

## ② データを取り込むためのプログラム

CALL #1, B, P

#1と書くと、BCDコードを使った入力サブルーチンに処理がわたされるわけです。

BASIC インタプリタ	#0 バイナリー 入力ルーチン	#1 BCD 入力ルーチン	#2 バイナリー 出力ルーチン	#3 BCD 出力ルーチン
-----------------	-----------------------	---------------------	-----------------------	---------------------

BASICに付加したI/Oプログラム

まず最初の#nこれは、サブルーチンの番号です。全部で4つあります。

#0…バイナリー入力  
#1…BCD入力  
#2…バイナリー出力  
#3…BCD出力

80 CALL #1, B, P  
90 PRINT B=:B

BASICのプログラムはこの変数Bでインプットデータを扱えるのです。

B=56

01010110

BCD2桁

argument1 これは入力値を入れる変数名を書きます。たとえば、これをBとすると、



P(0)… エラーの内容が入る

P(1)… デバイス No.

基板の0か1

P(2)… ポート No. に1加えたもの(1, 2, 3, 4)

P(3)… データ数(この場合1)

P(4)… 正論理 / 負論理(この場合1)

P(5)… ストローブ / レベル(この場合0)

P(6)… インターバル(この場合必要なし)

P(7)… 自動開始 / EXTACK開始  
(この場合必要なし)

このパラメータはCALL文実行の前にそれぞれの要素に必要な数値をセットしておきます。

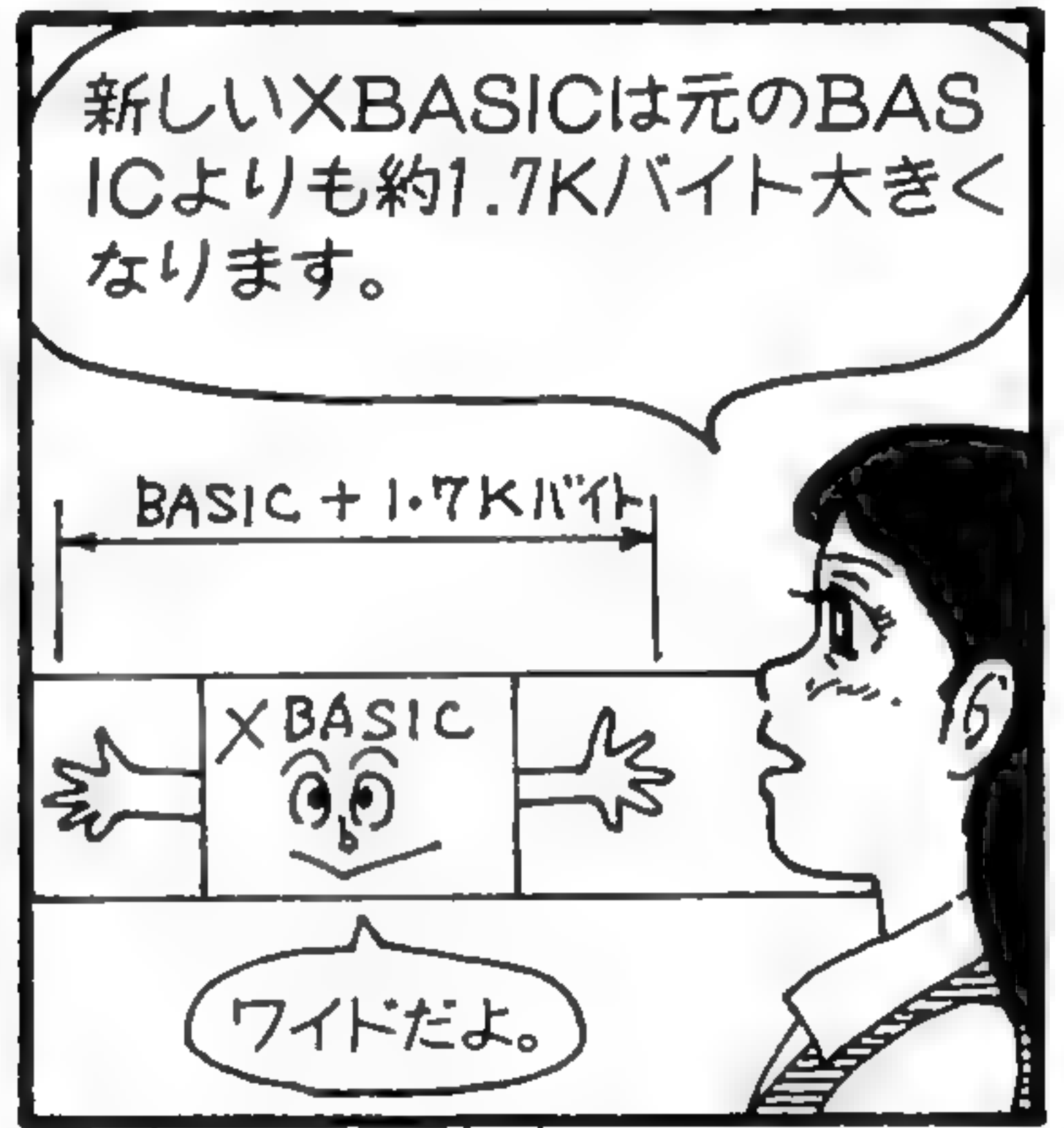
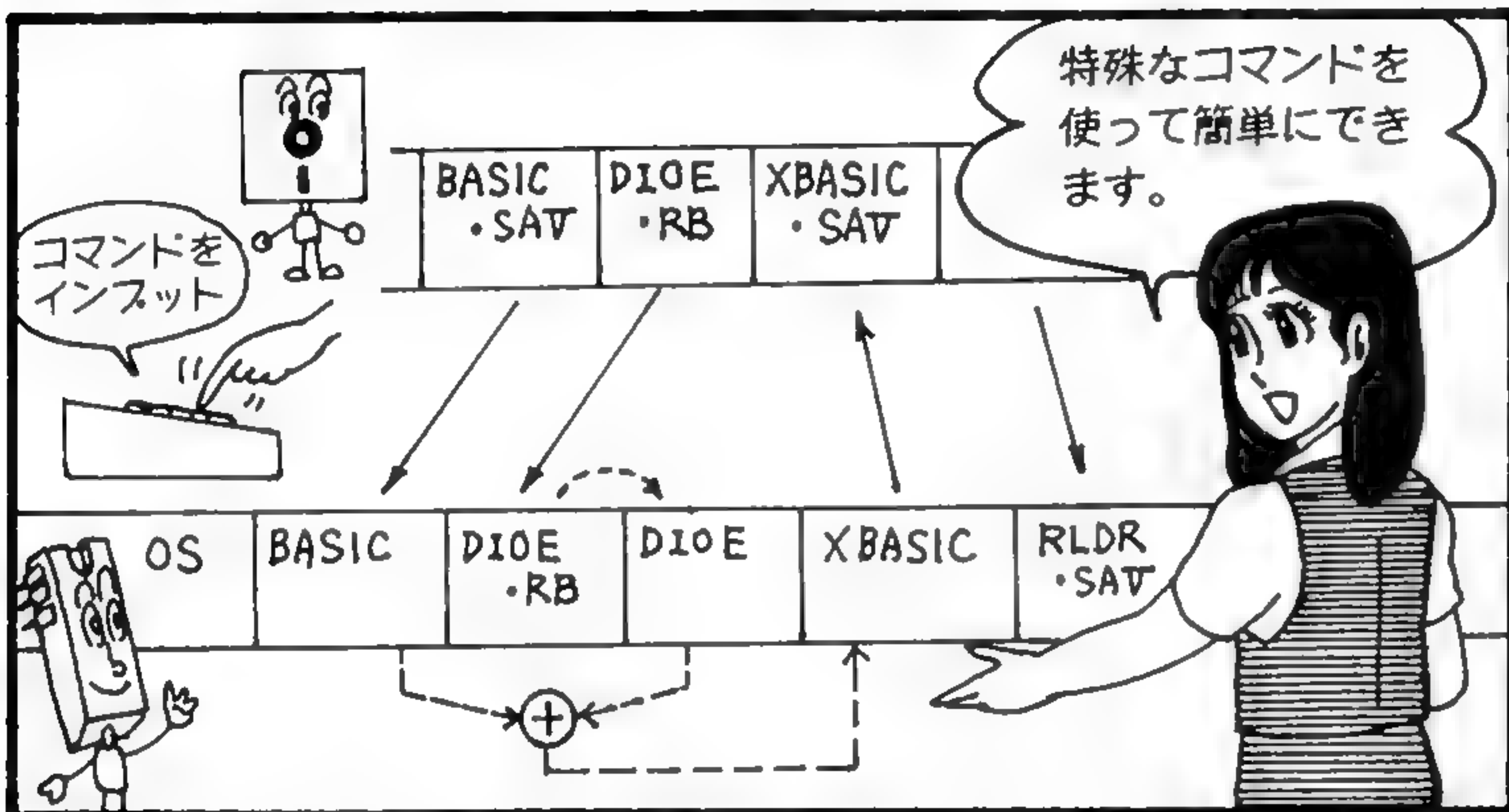
argument 2 これはパラメータの設定です。一次元配列を使ってプログラムします。

DIM P(10)

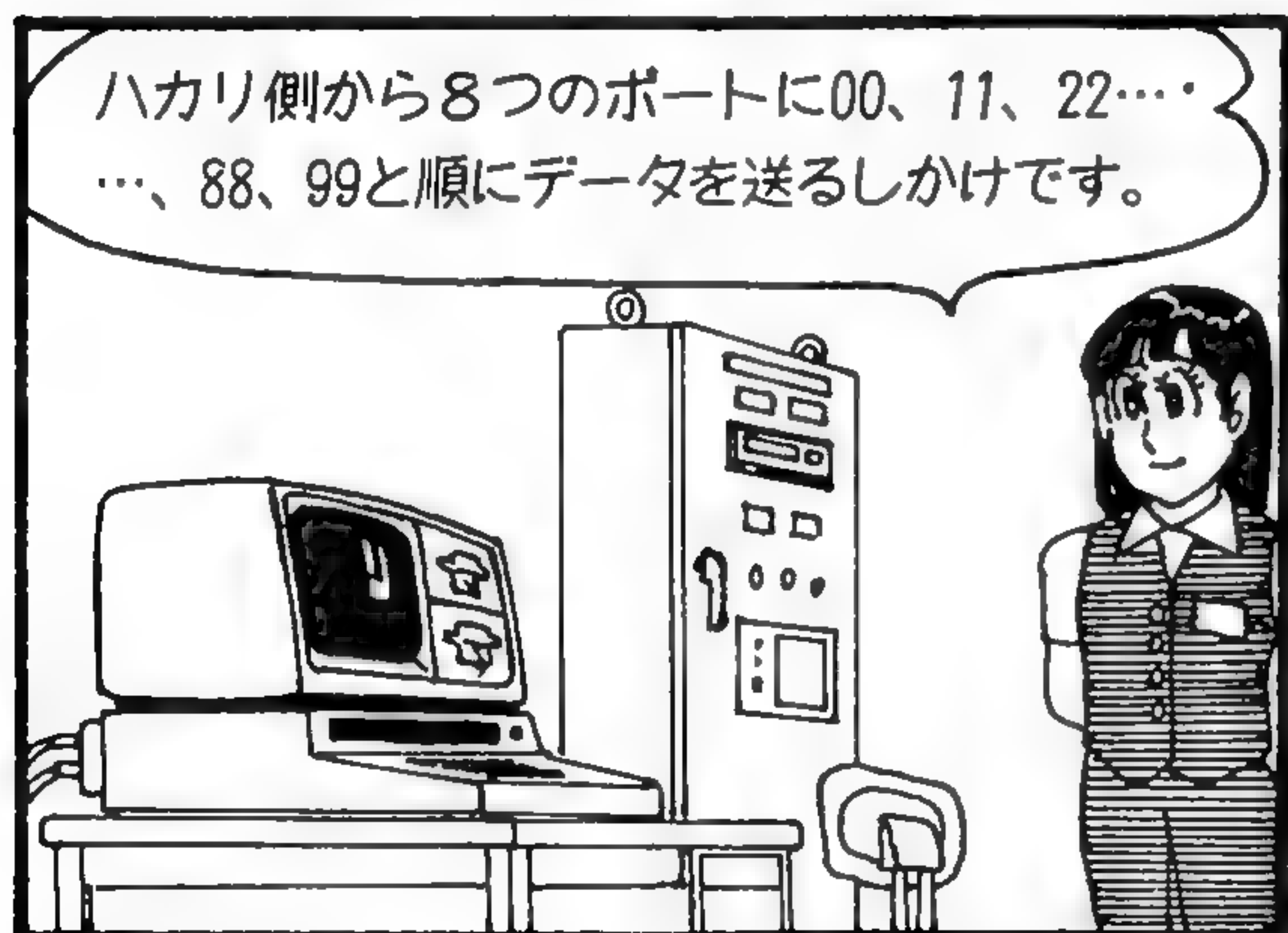
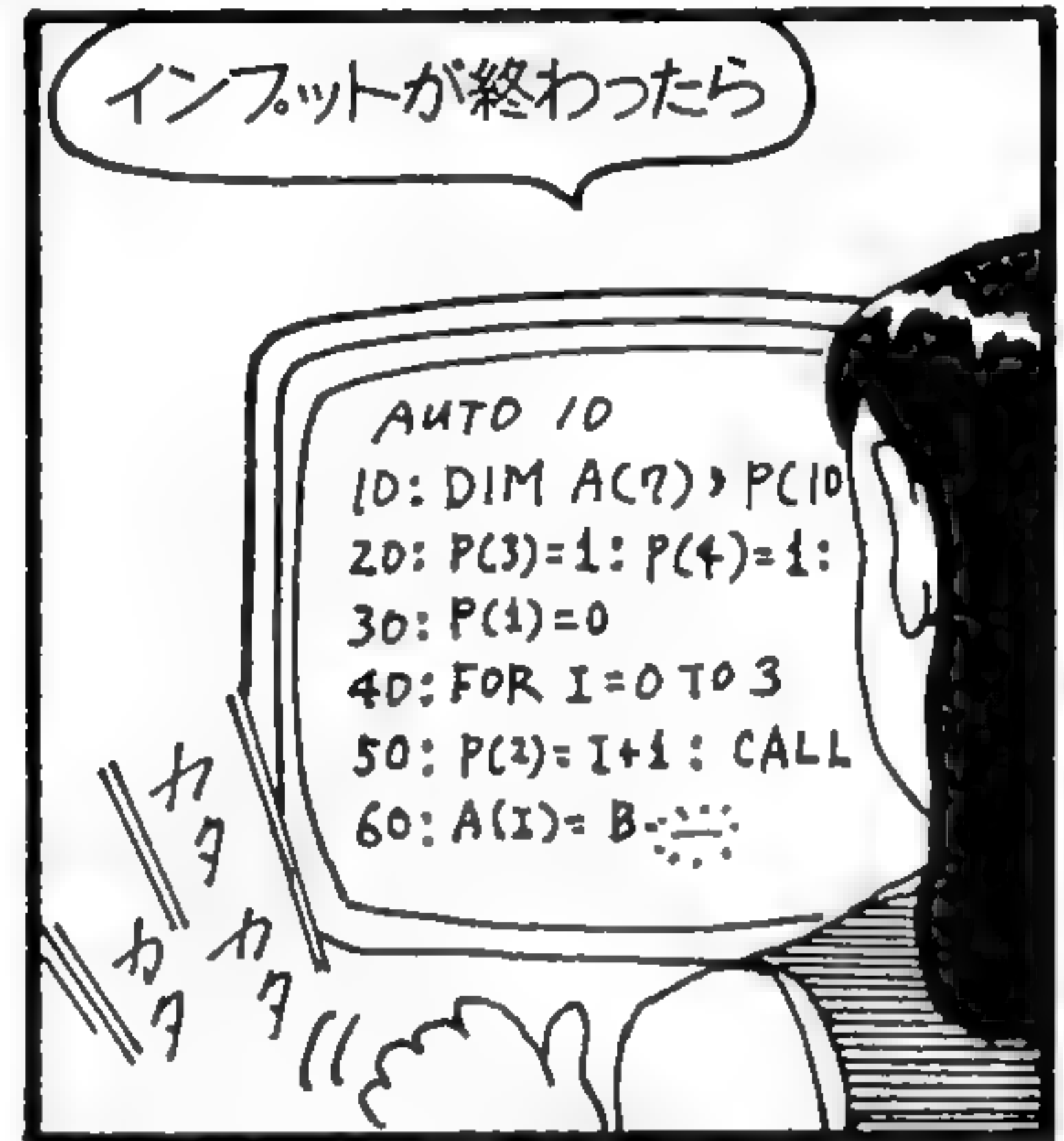
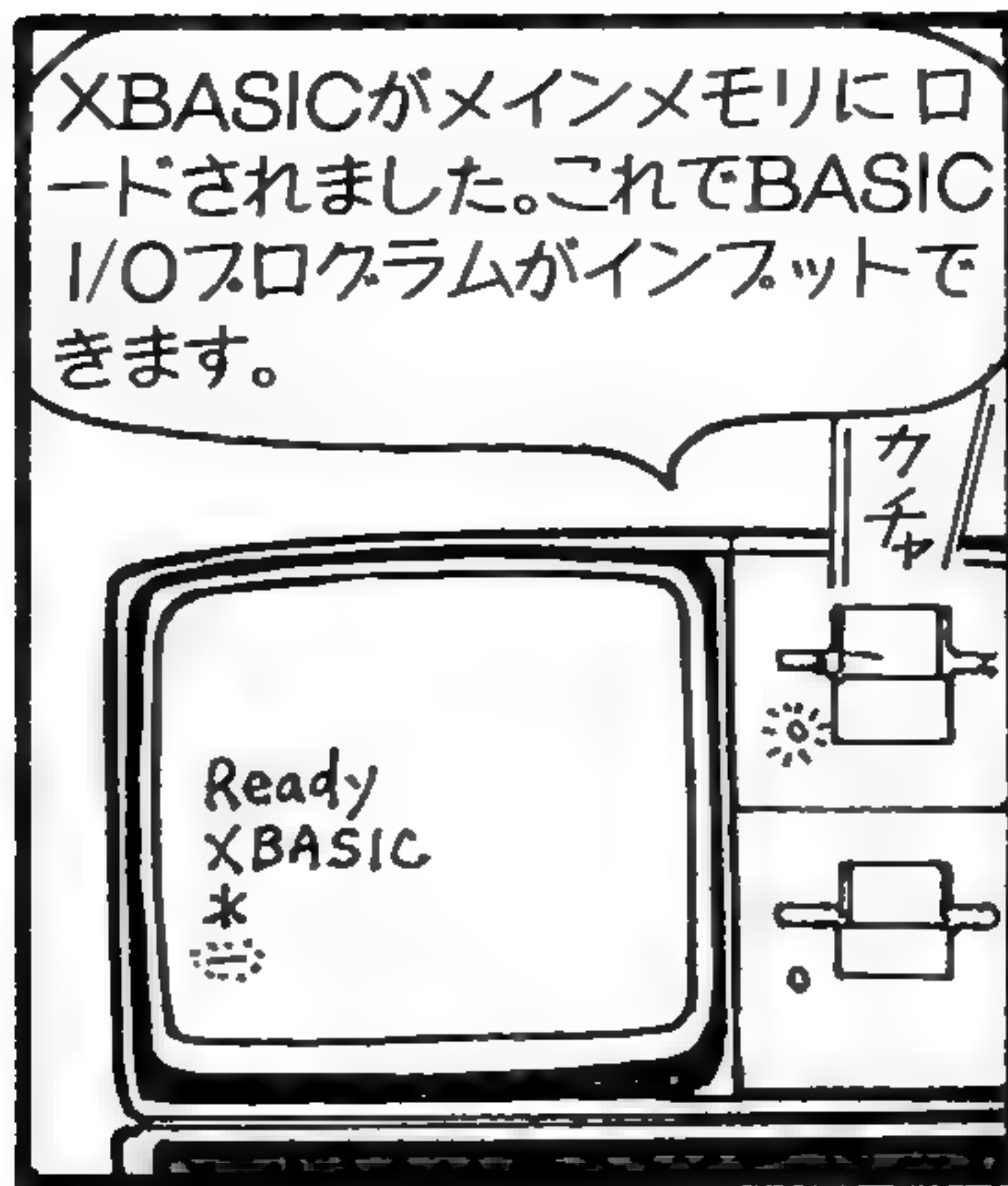




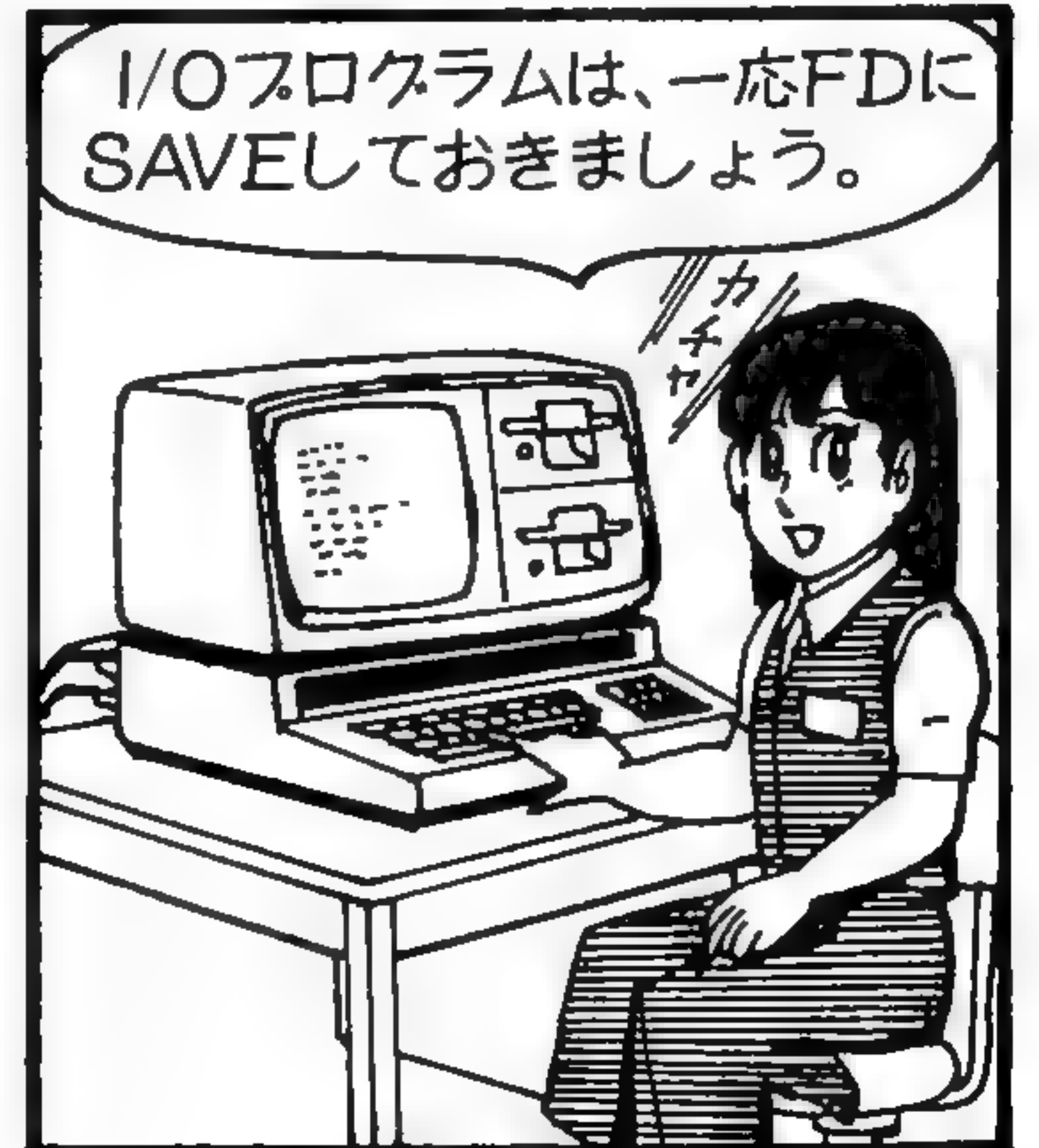
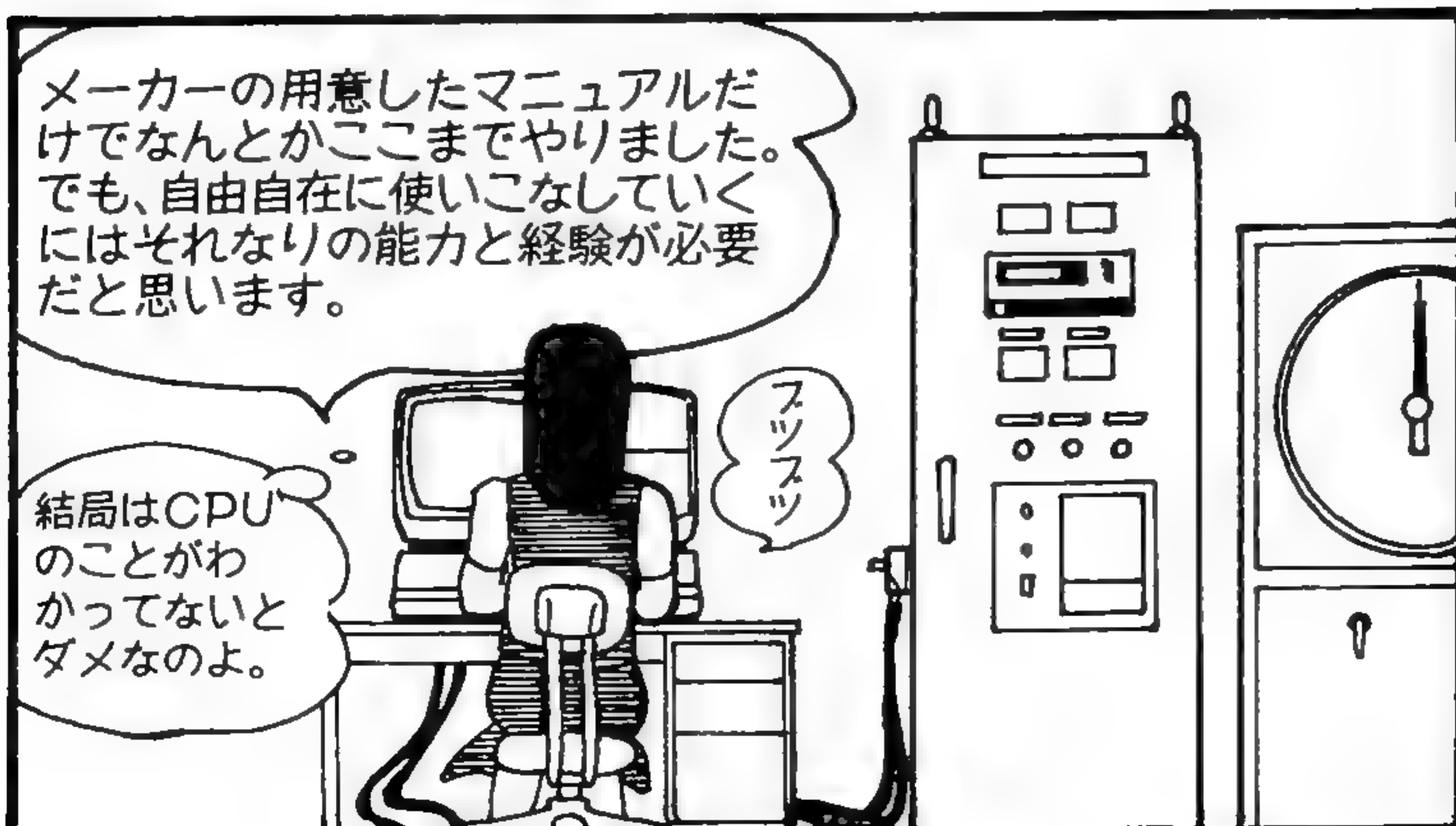
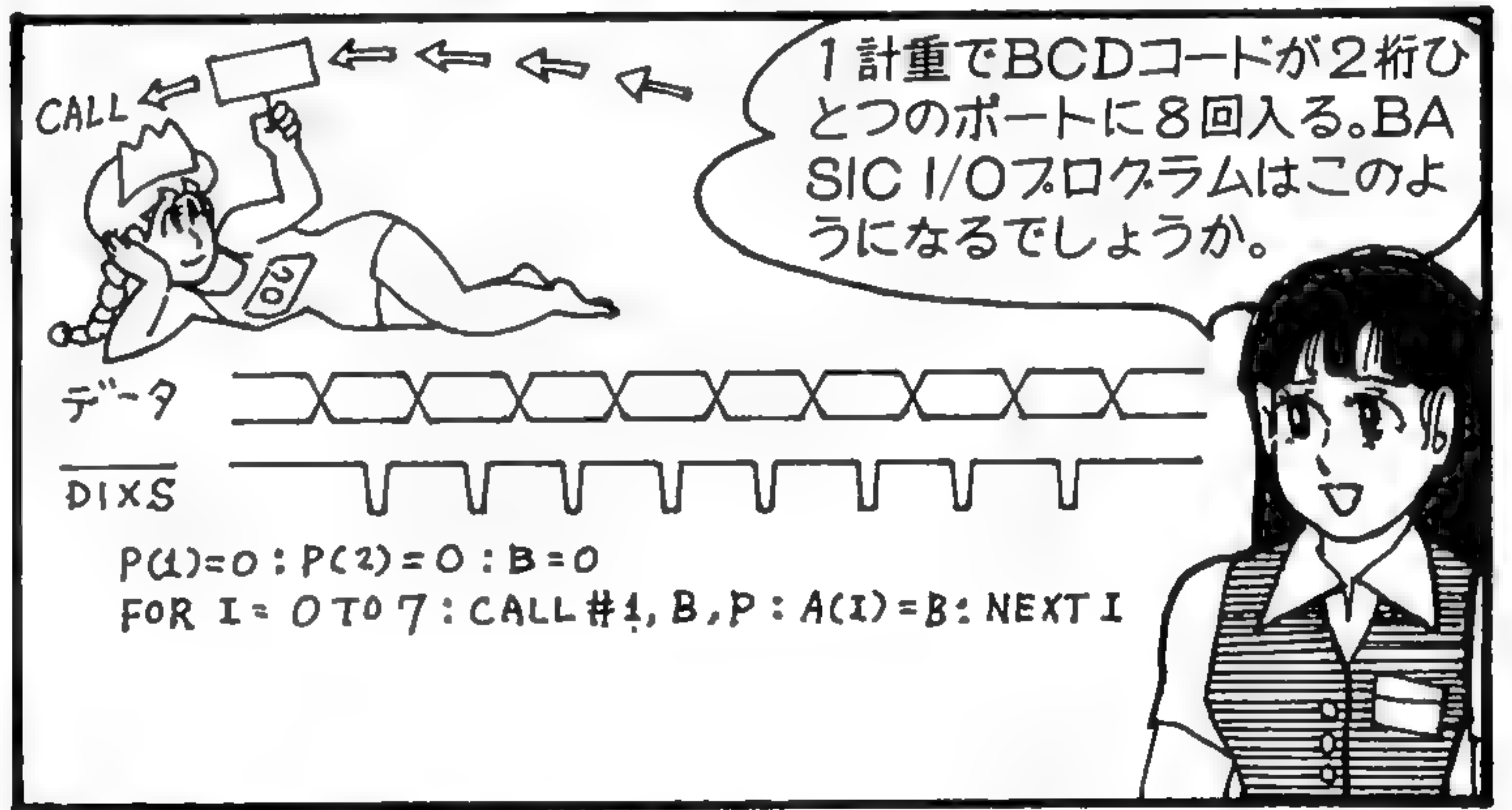
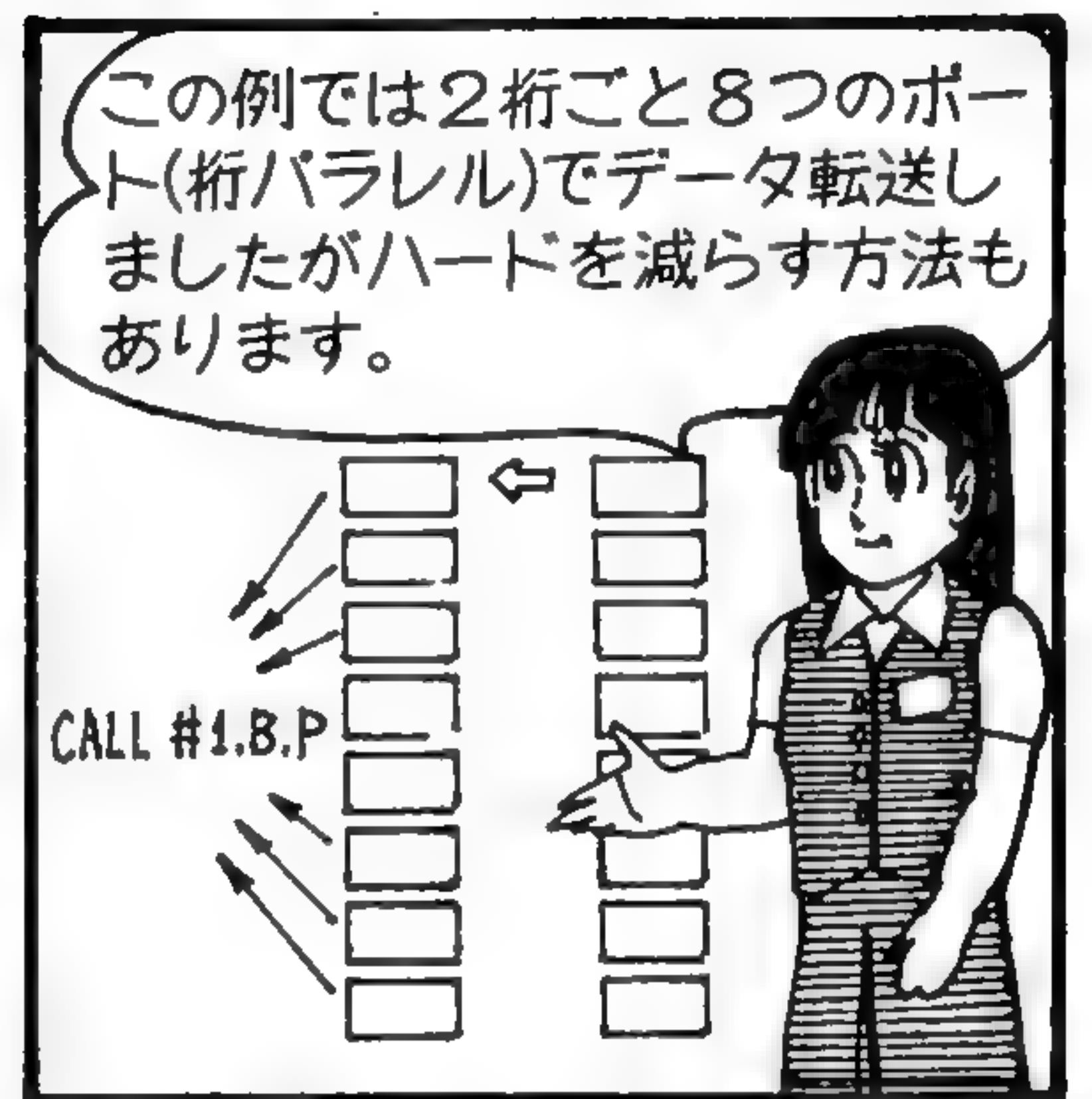
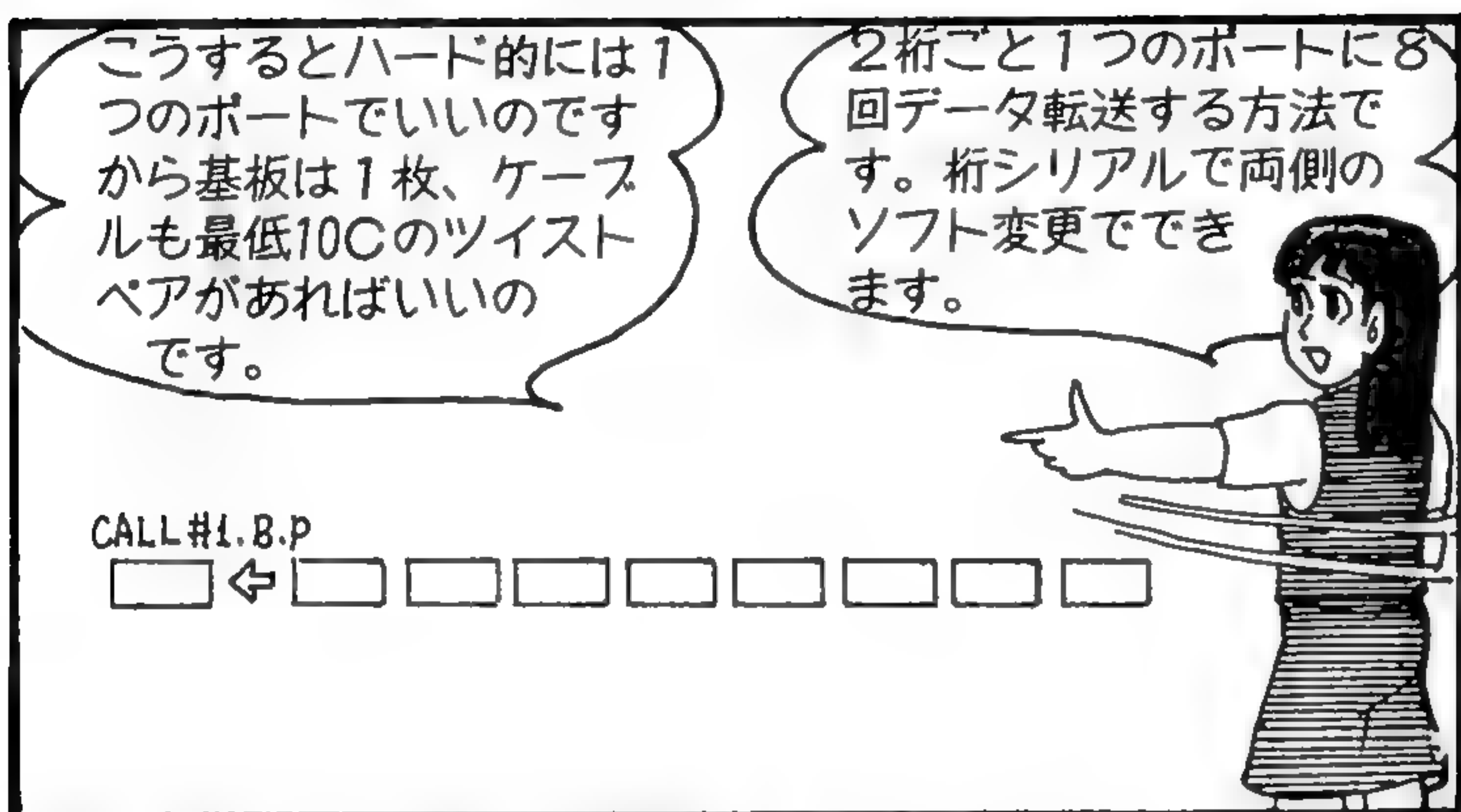
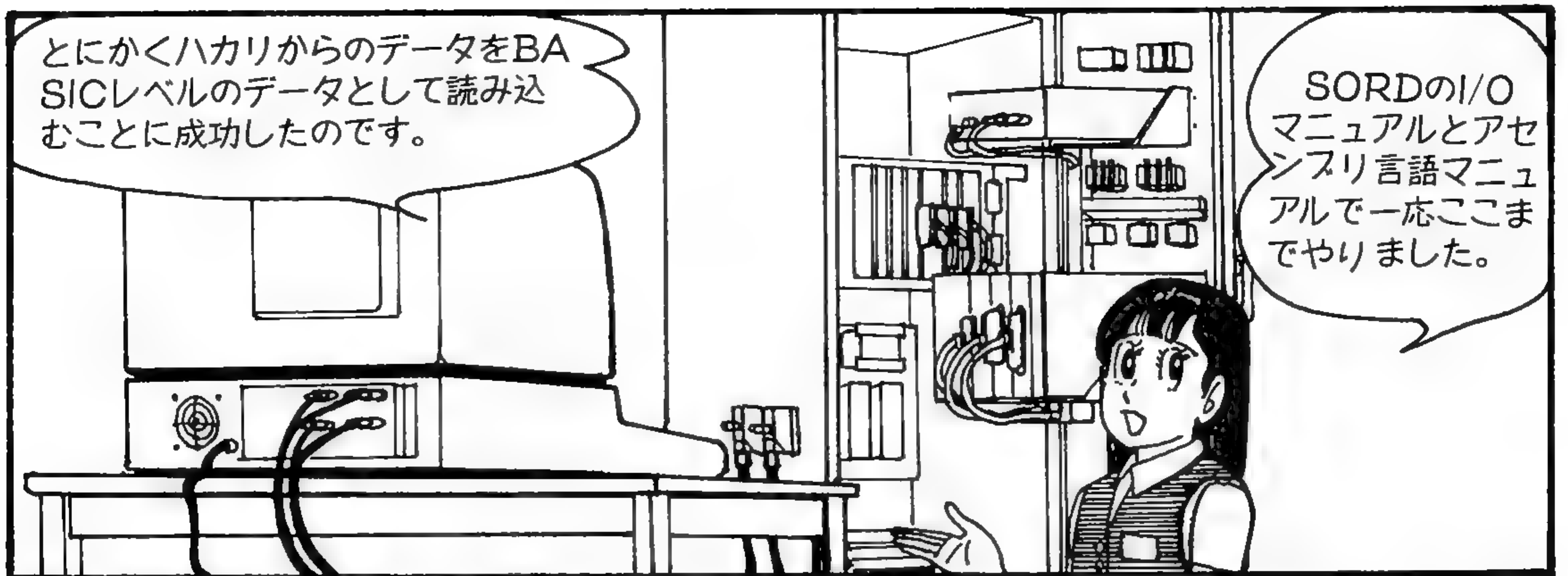






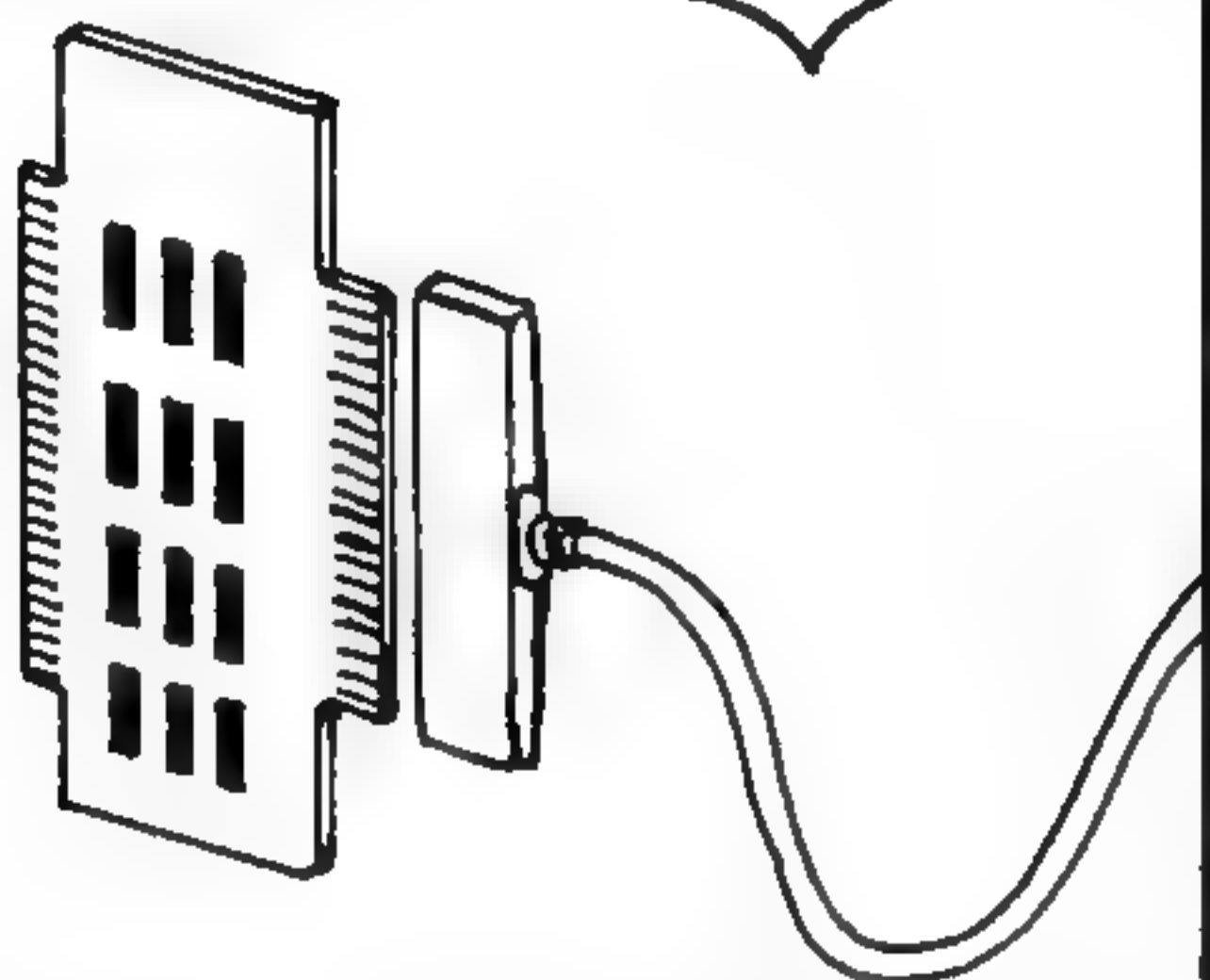




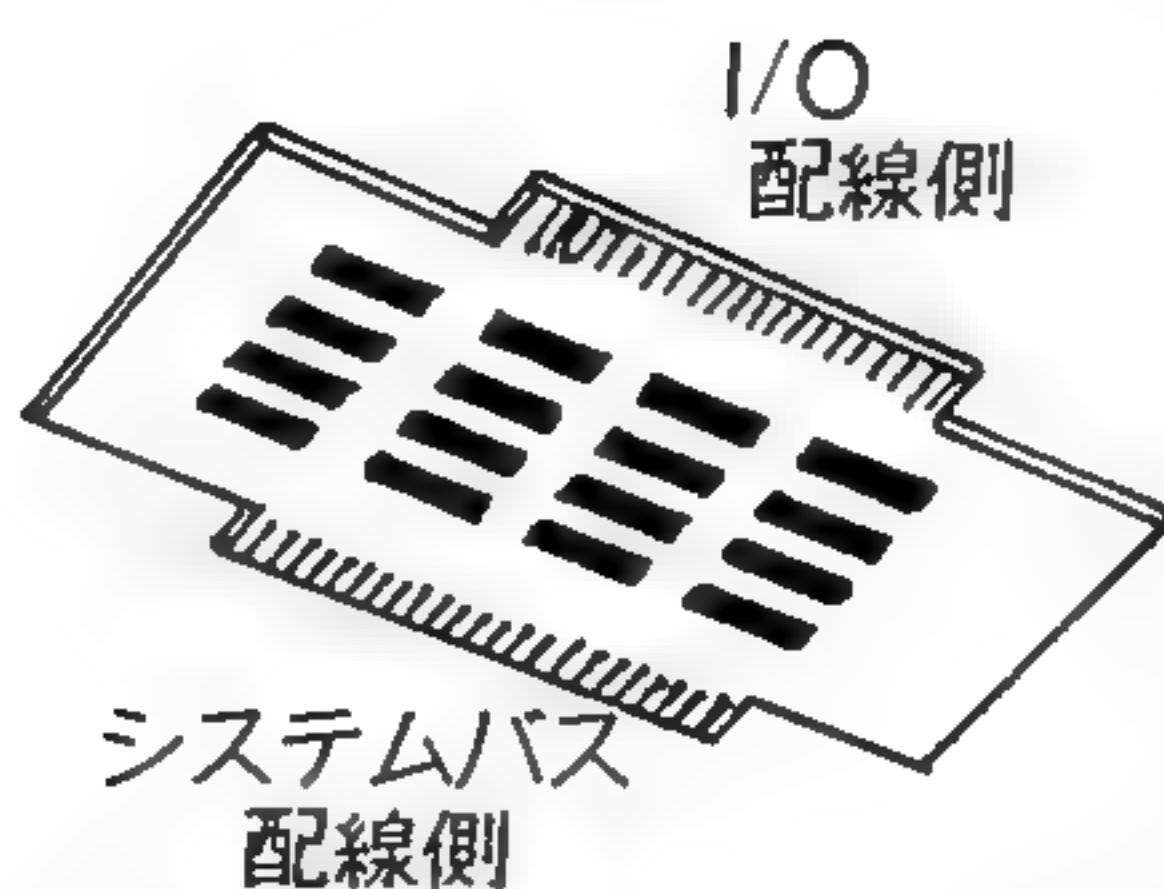




こうすると制御盤内のラックに組み込んだ時、システムが単純化できるのです。



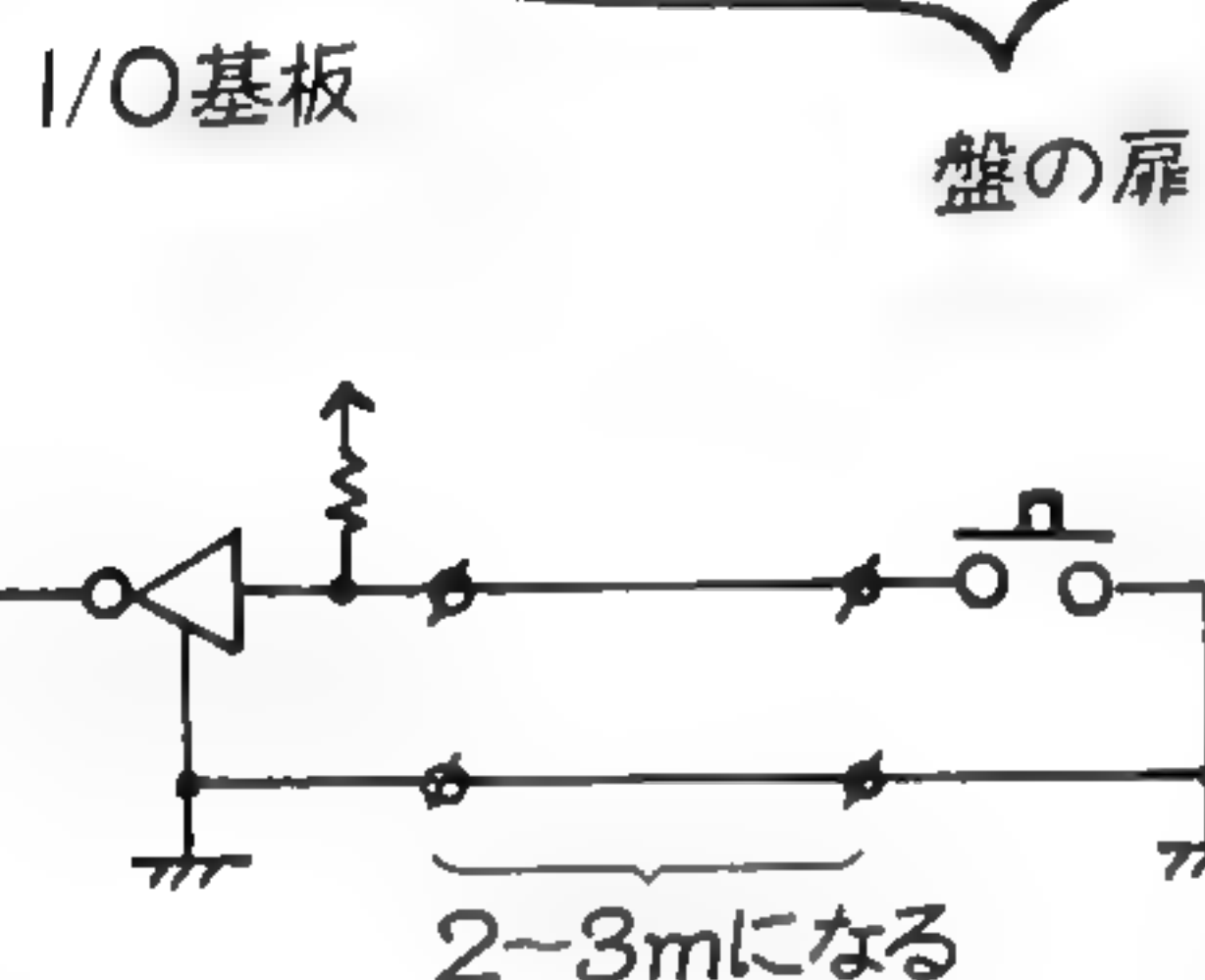
I/O基板にはシステムバス用コネクタの反対側にI/Oのための配線用コネクタがあるものが便利です。



### ③ I/O回路のノイズ対策

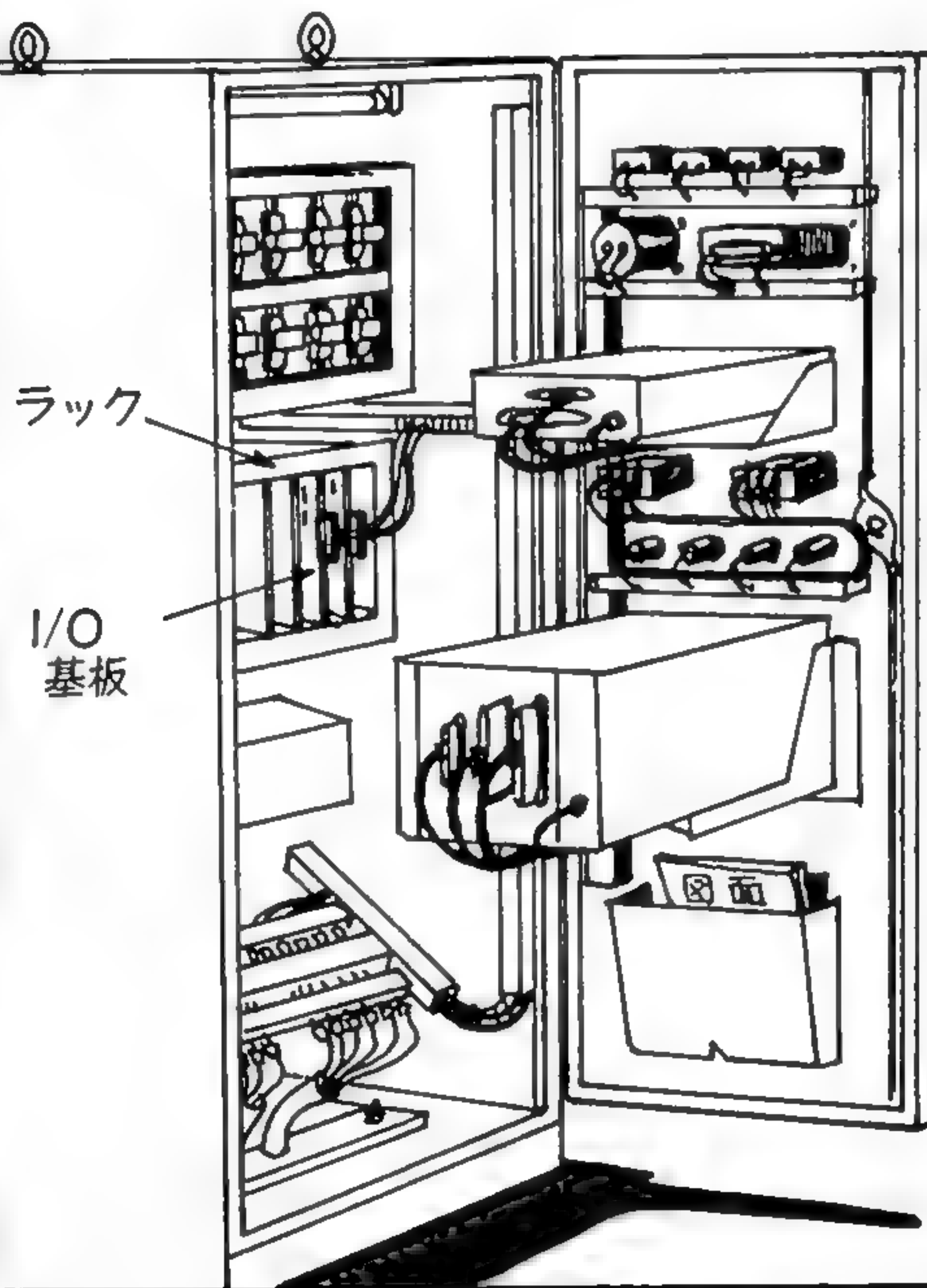


これはボタンによるインプットの回路例ですが、この図面では現場でどう配線されるかわかりません。

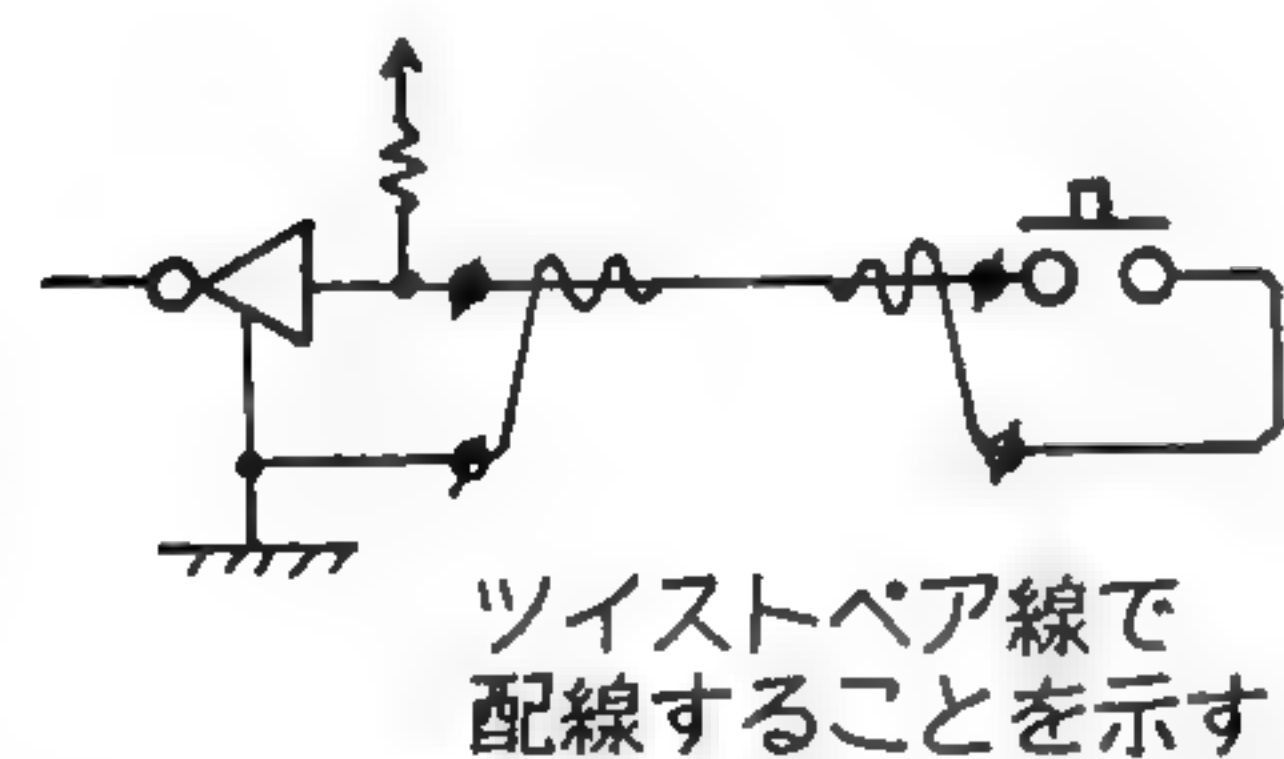


つまり、応答速度やノイズへの対策が必要になってくるのです。

ここで実用上どんな問題があるかといいますと、I/O基板から出た信号線はほとんど盤の正面のボタンや表示器に出て行くため配線が長くなるということです。

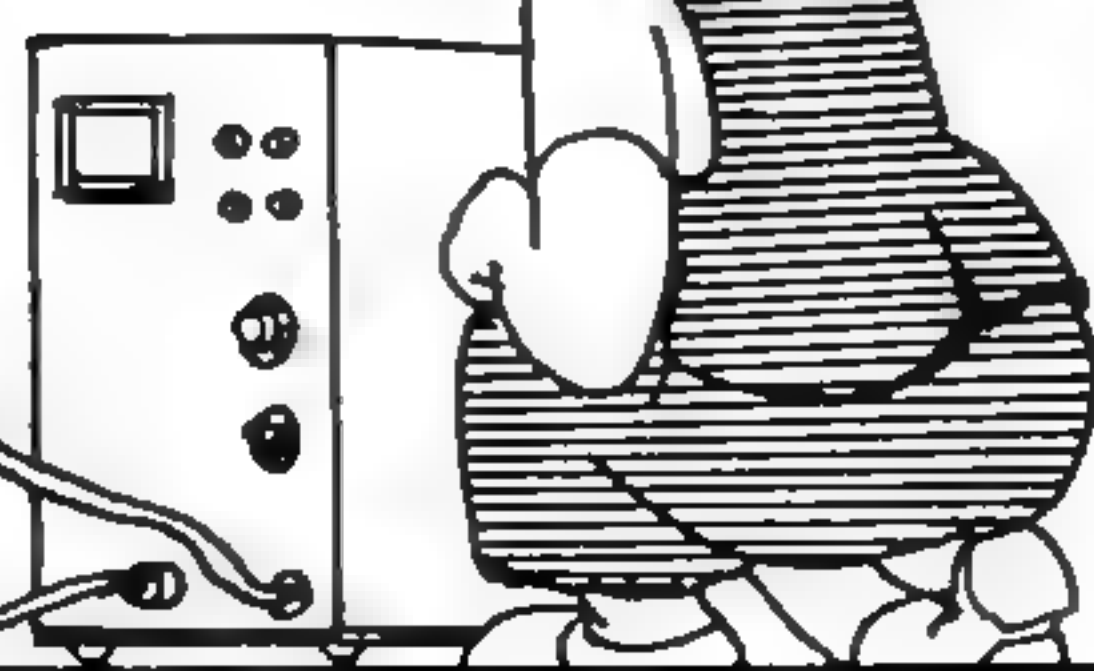
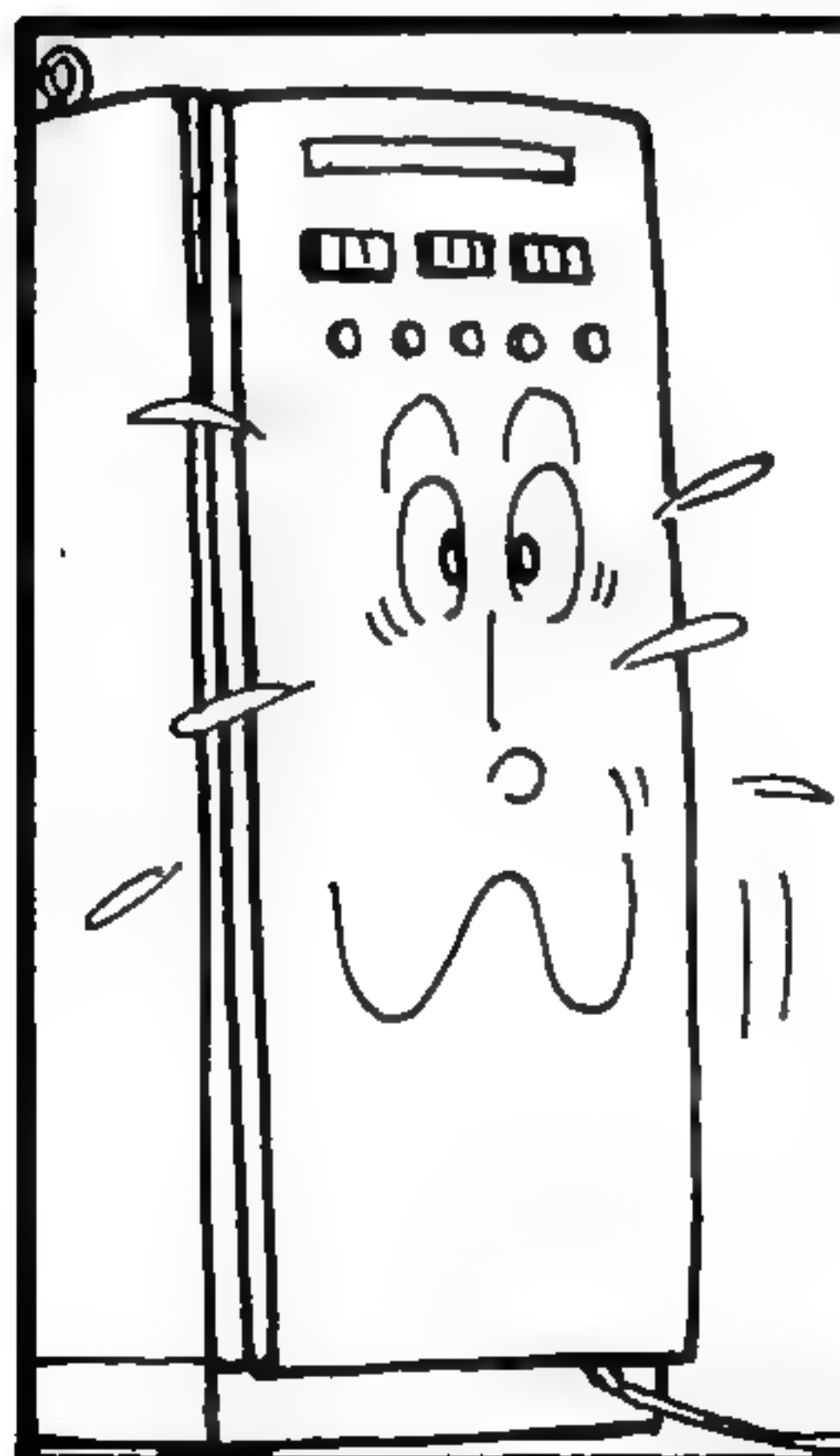


図面にはこのようにツイストペア線を用いることを明示し、コモン側はI/O基板のついているラックのアースまで引っ張ってくる必要があります。

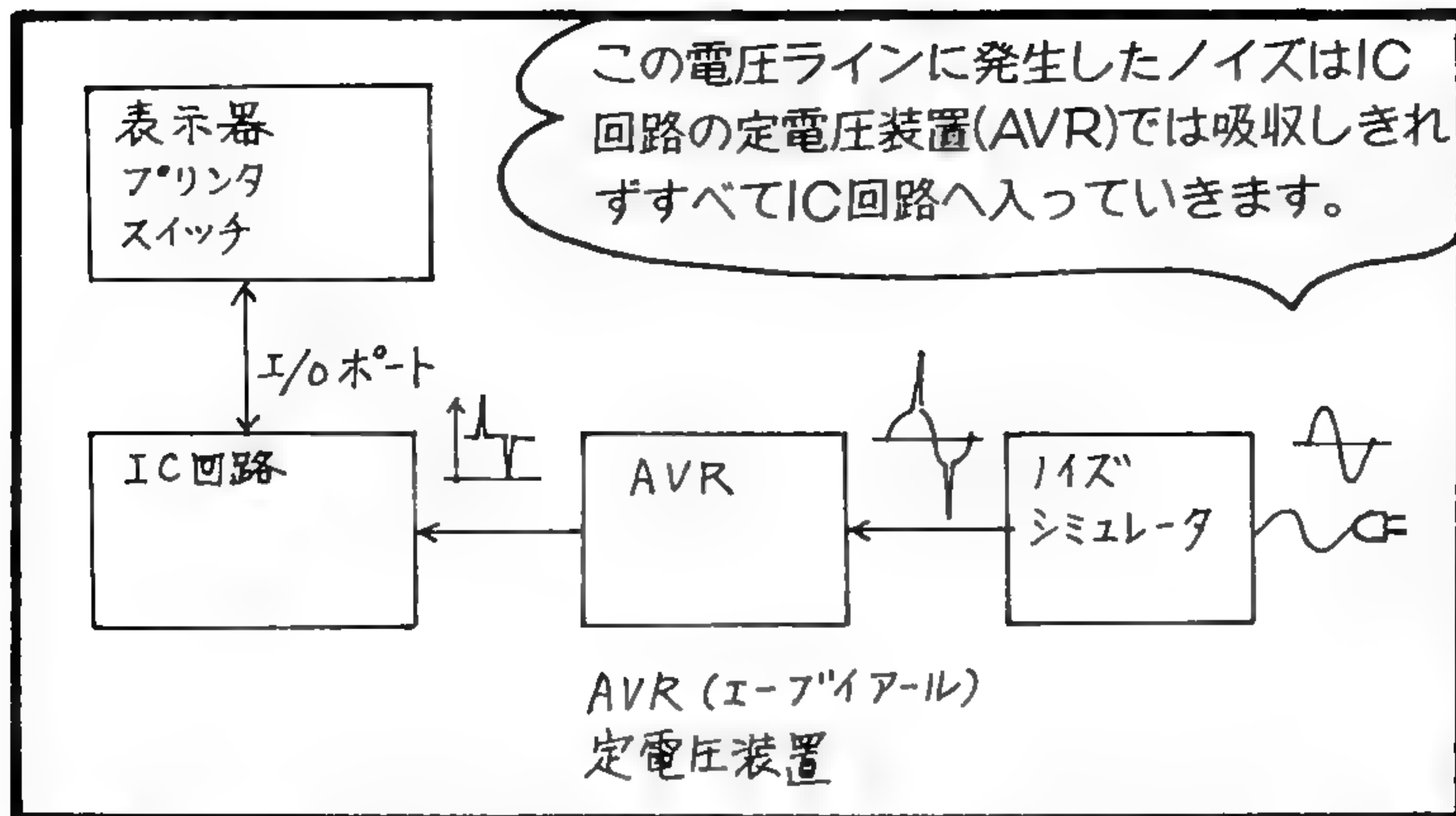


ノイズテストにはノイズシミュレータが使われます。電源ラインに500Vとか1000Vのサージ電圧を発生させる装置です。ノイズ対策のない回路の場合はひとたまりもなく誤動作します。

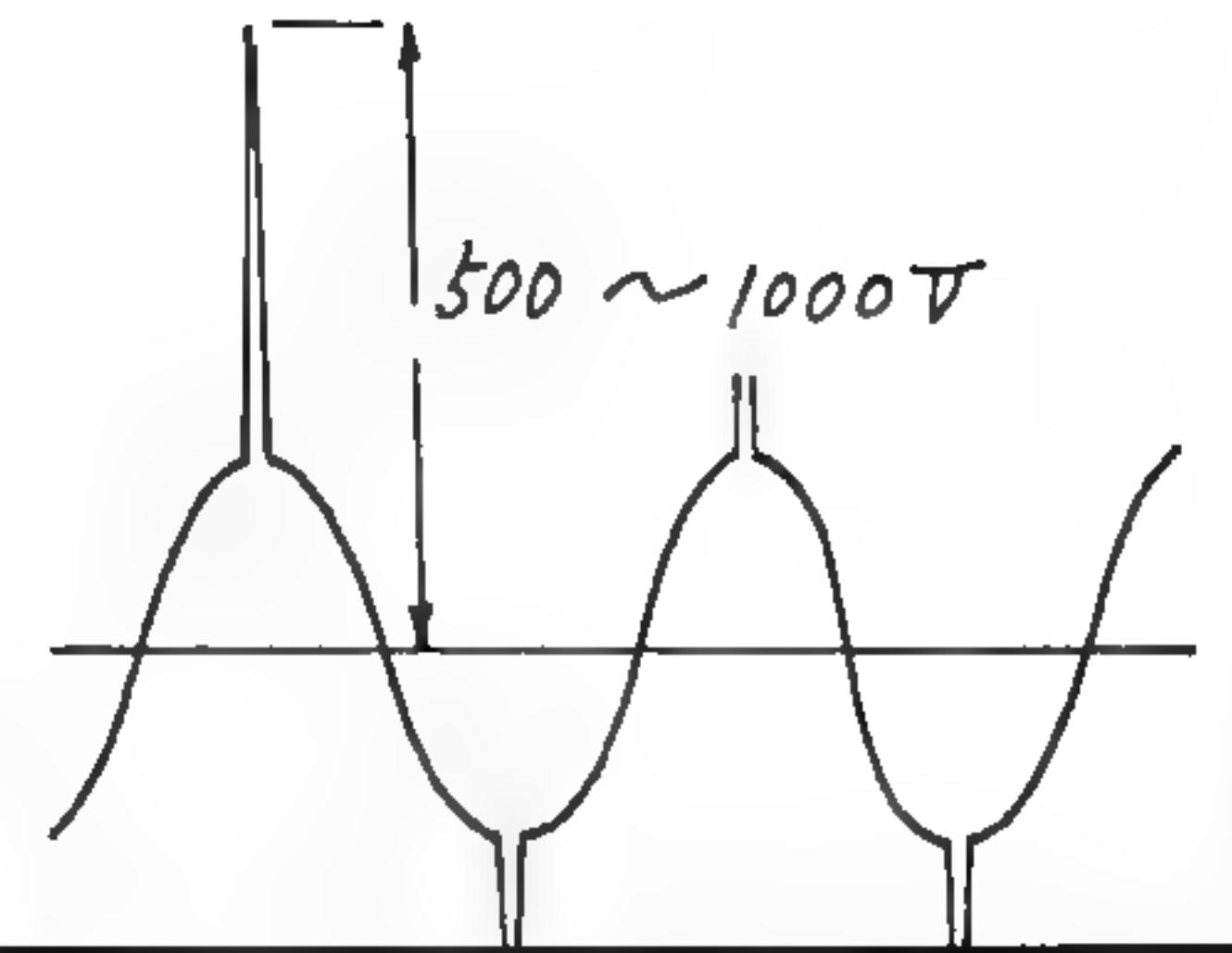
このツイストペア線を用いるとコモンモードノイズに対してかなりの効果が期待できます。効果はケースバイケースであんまり効かない場合もあります。



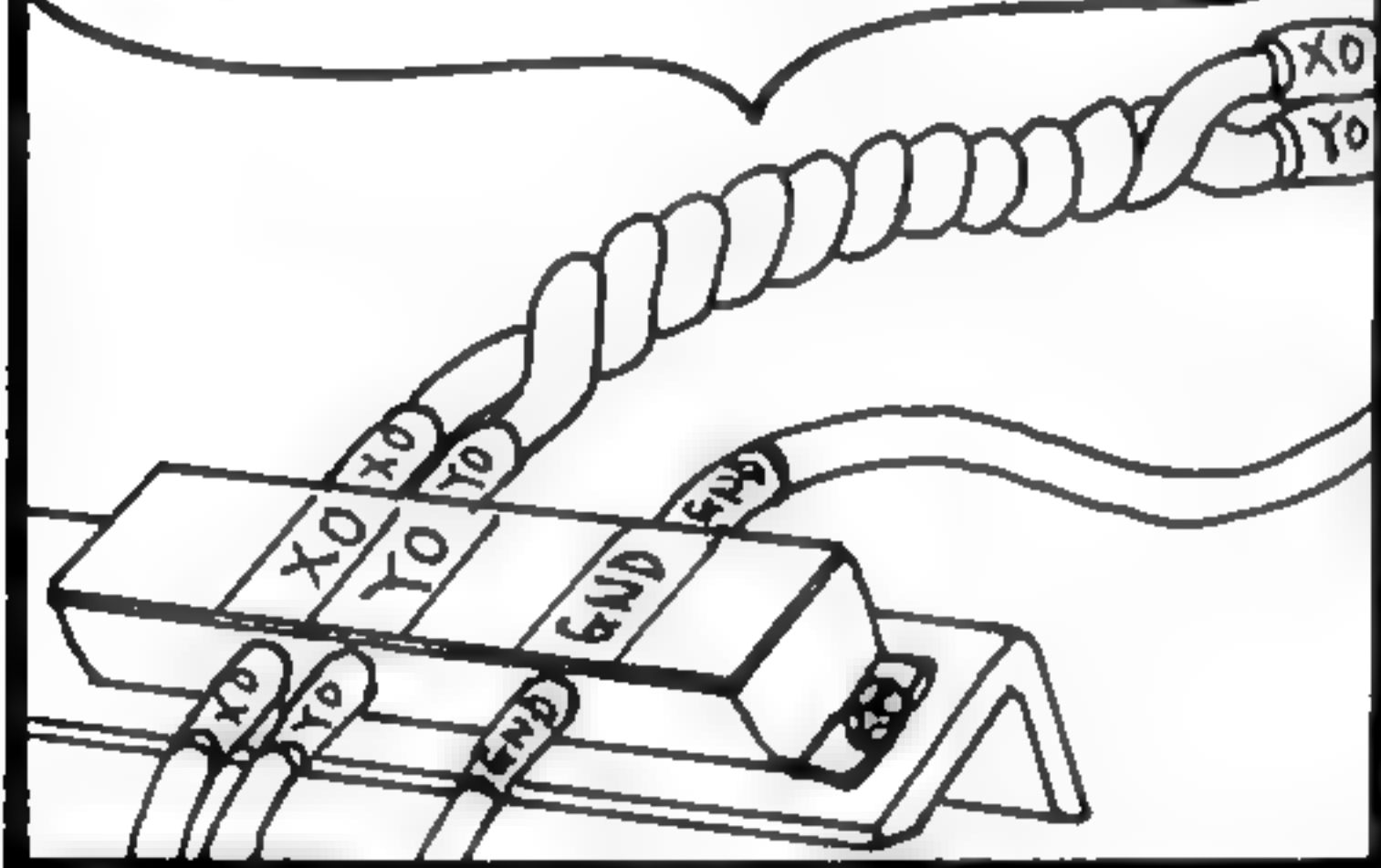




ノイズシミュレータを経由した電源には図のようなサージ電圧が発生しています。



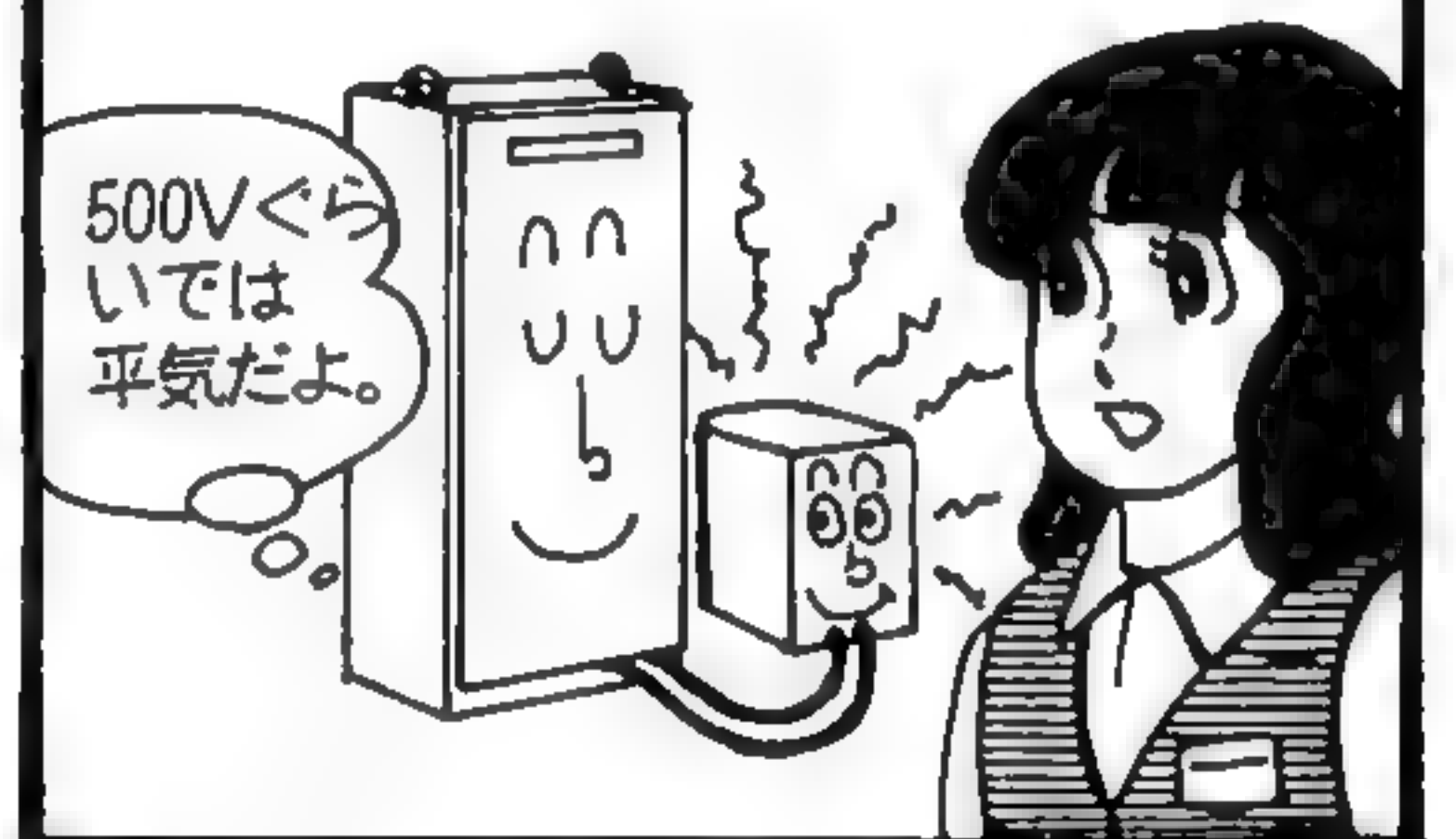
まず、信号線のツイストペア化の他に、電源ラインについてもツイストすることによって同一ダクトに収納される信号線への影響を少なくするようにします。



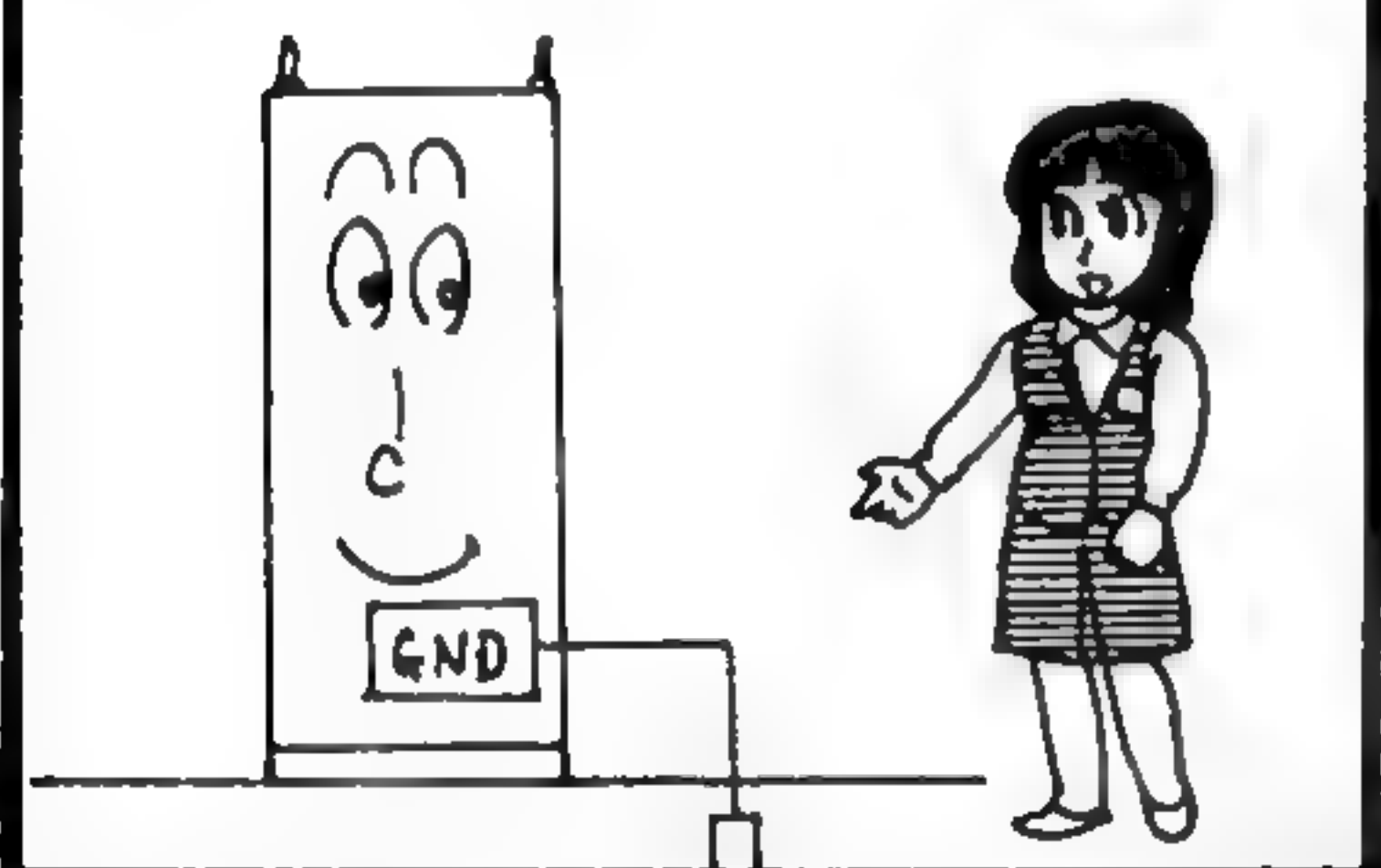
フリップフロップの反転やCPUの暴走と思われる現象が起こったら、



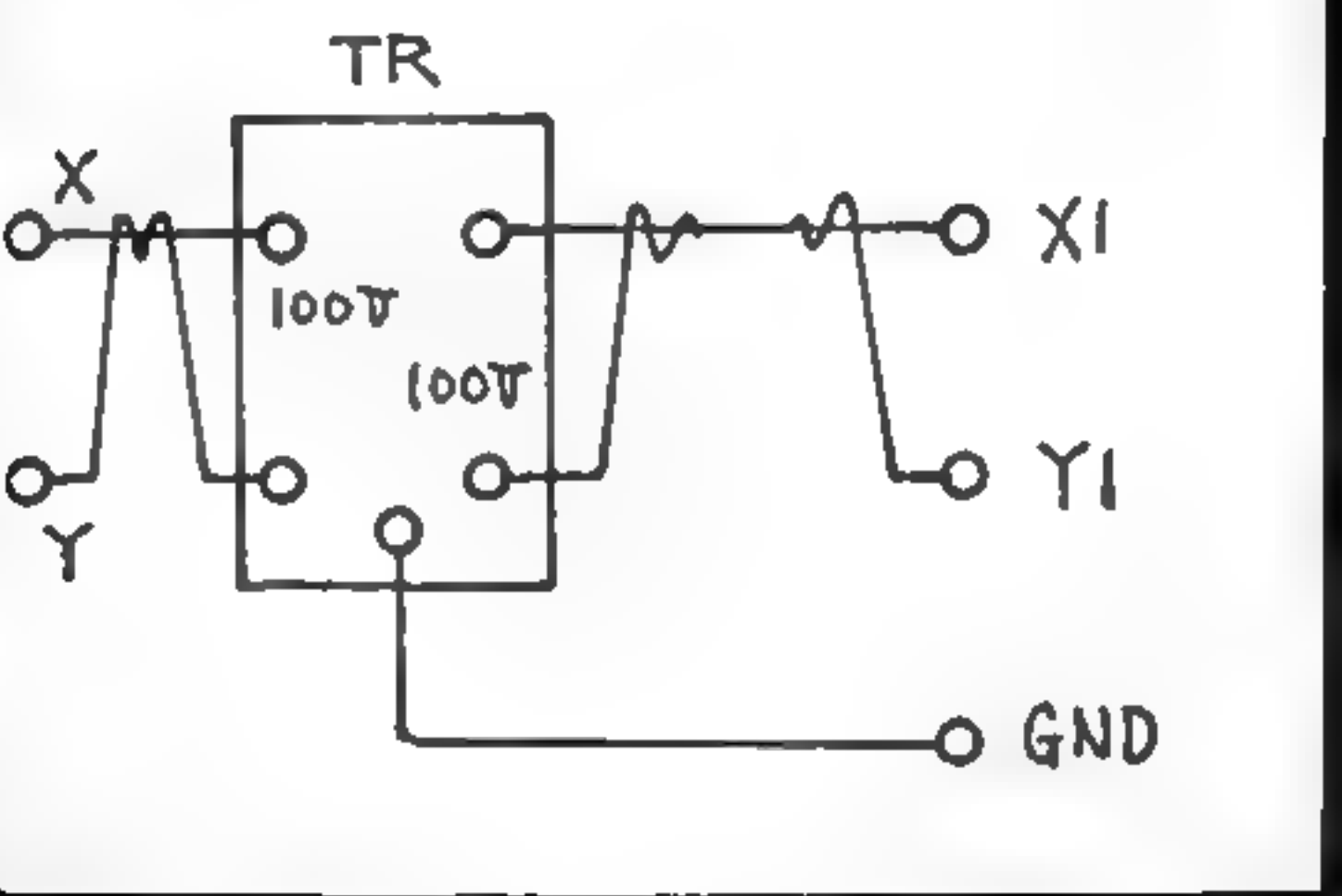
したがって、このノイズシミュレータを使ってテストしておけば電源ラインのノイズ対策ができるわけです。



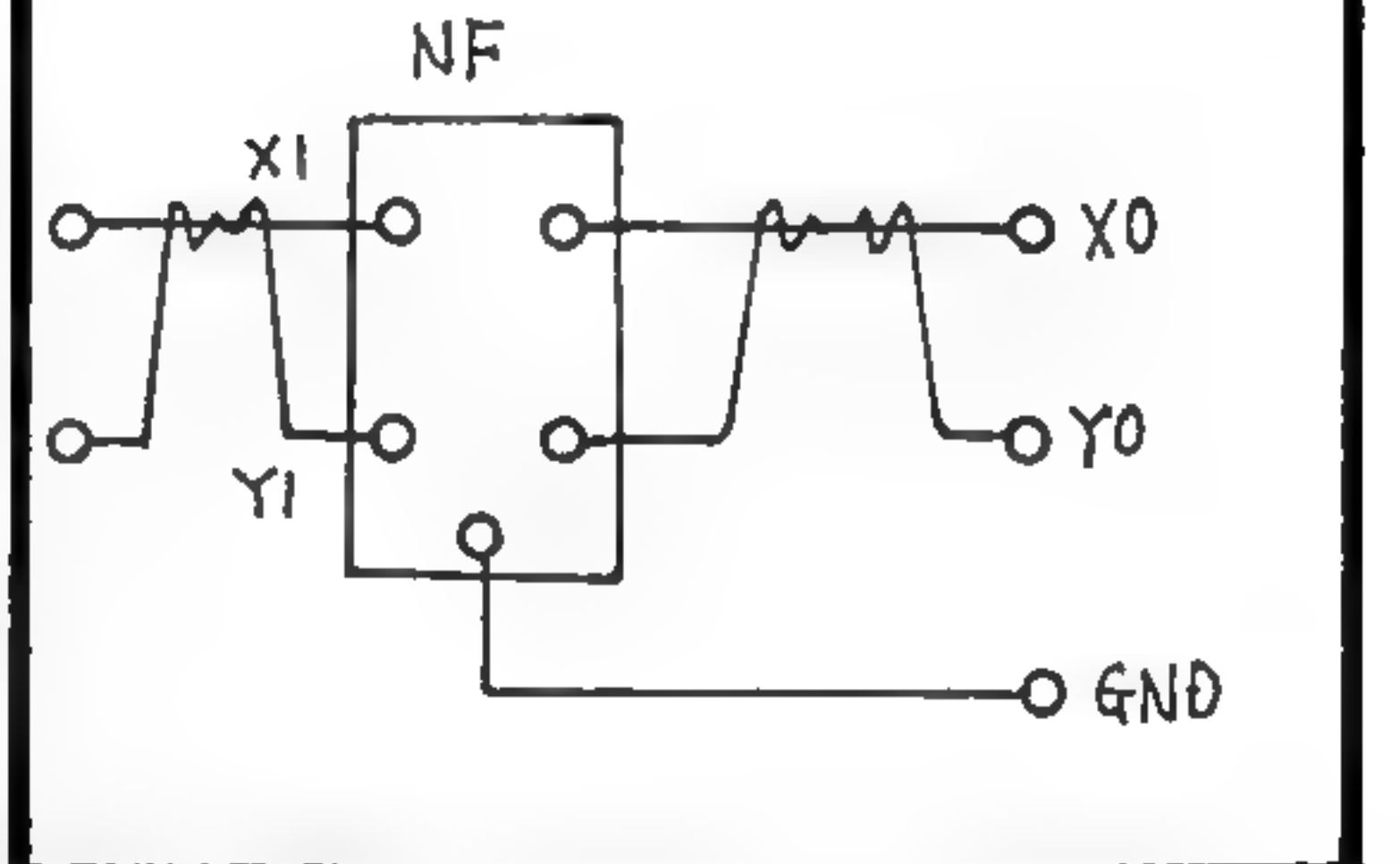
またこれらのノイズ対策機器のアースはきちっと接地されていなければ効果はありません。



さらにシールド付トランスをつけるといっそう効果的ですがコスト高となります。



また、盤に入ってすぐの位置にノイズフィルタをつけるのも効果があります。



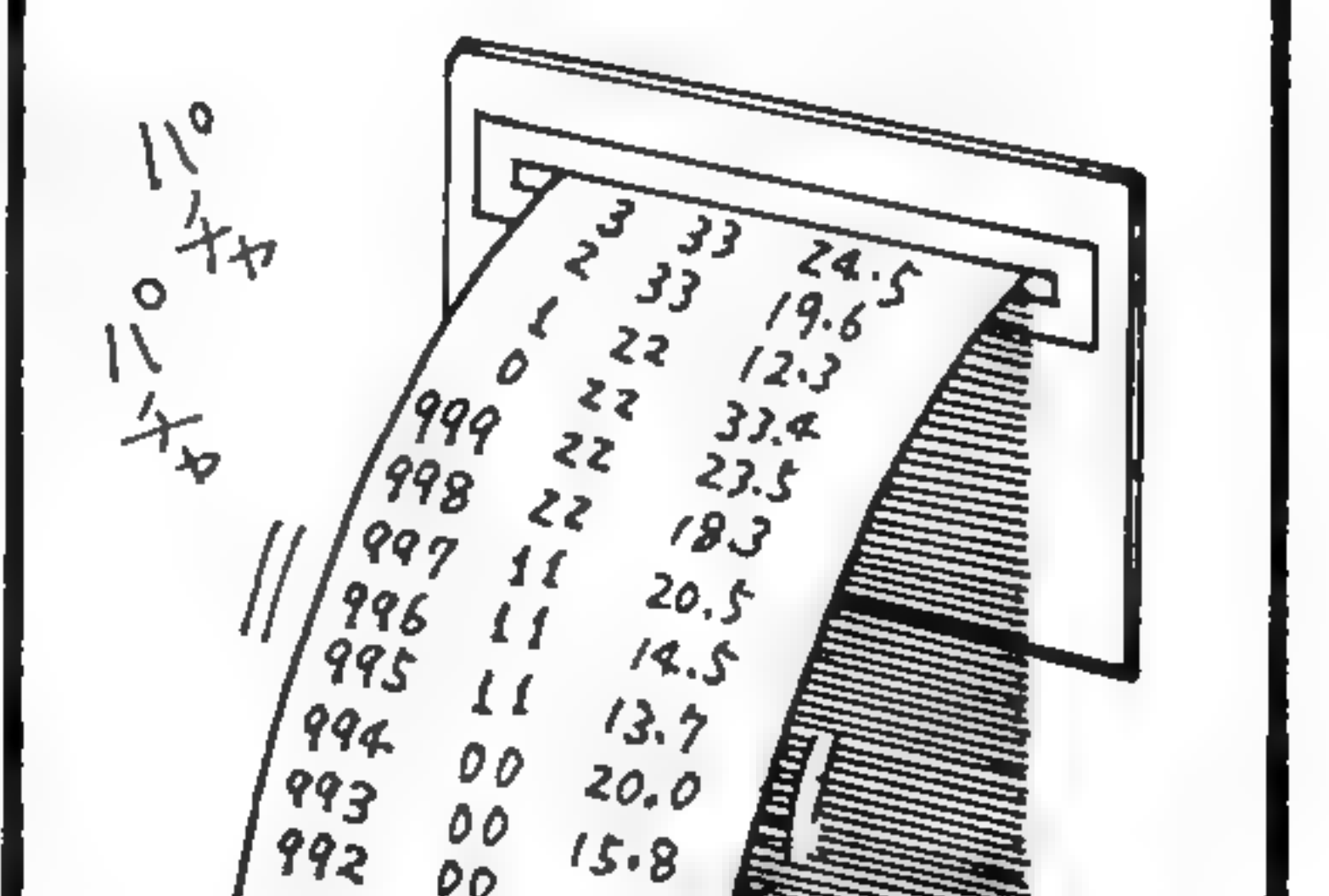
設置したあとで単発的に誤動作し、設備担当者やメーカーを泣かせるということもあるのです。



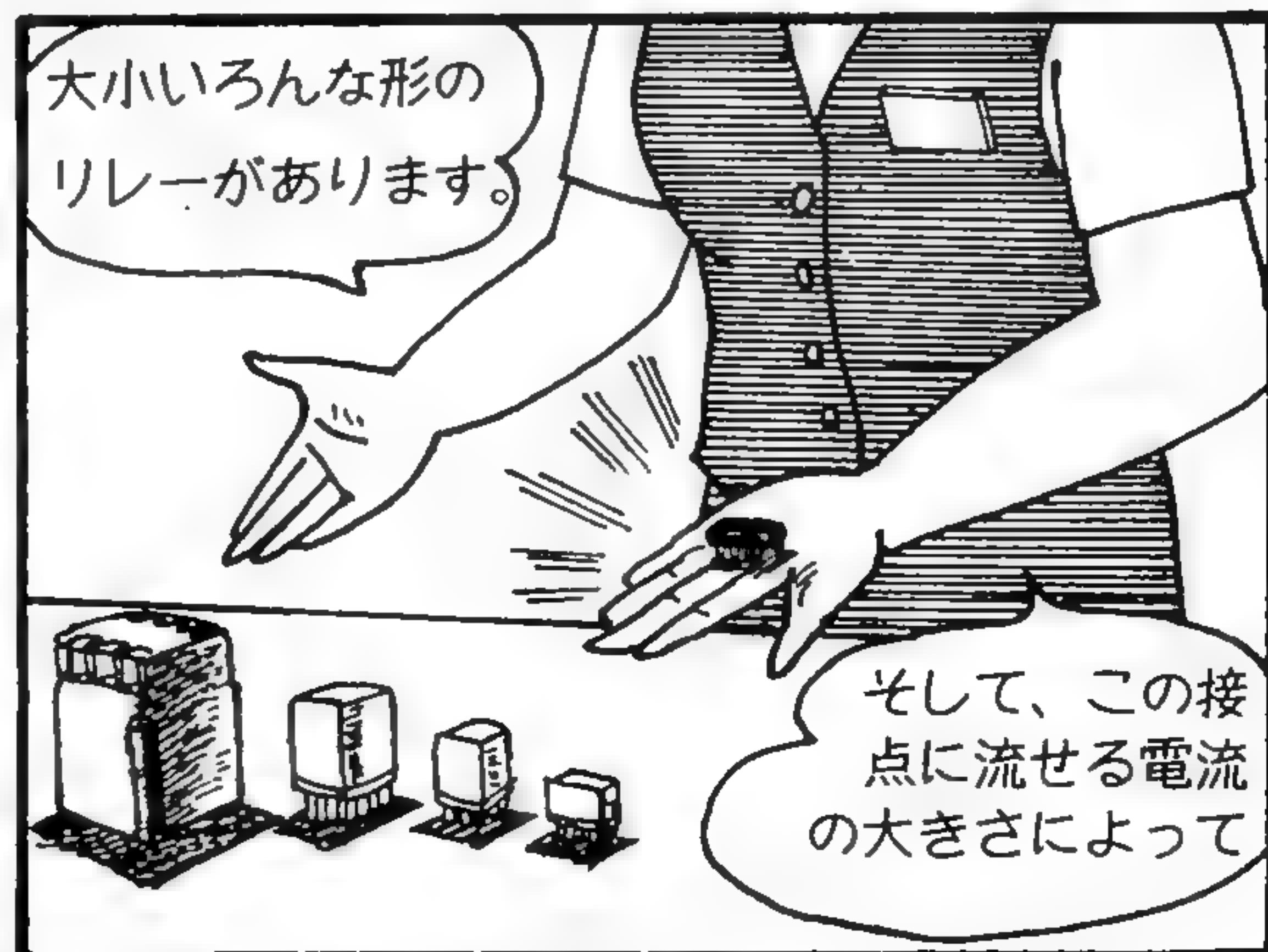
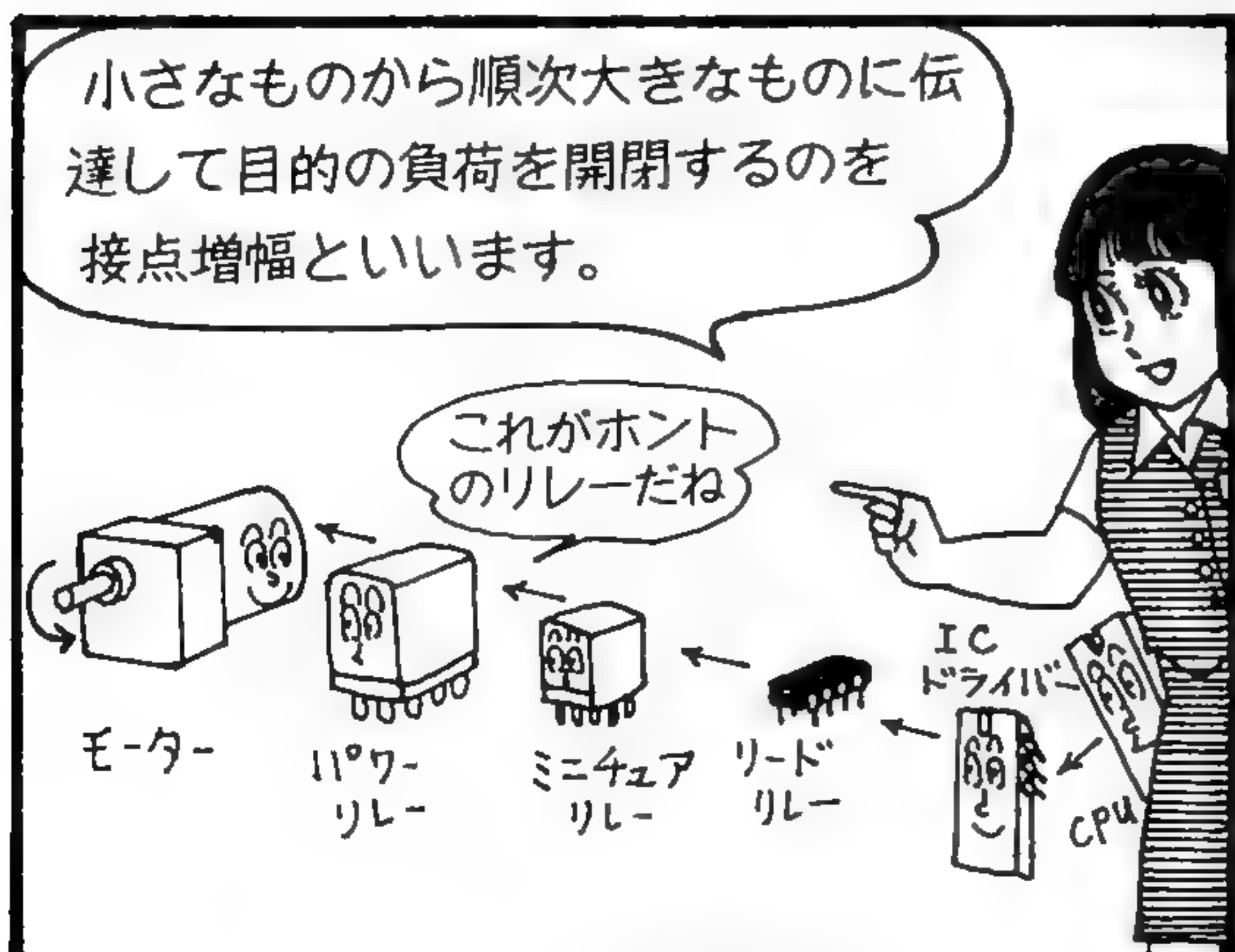
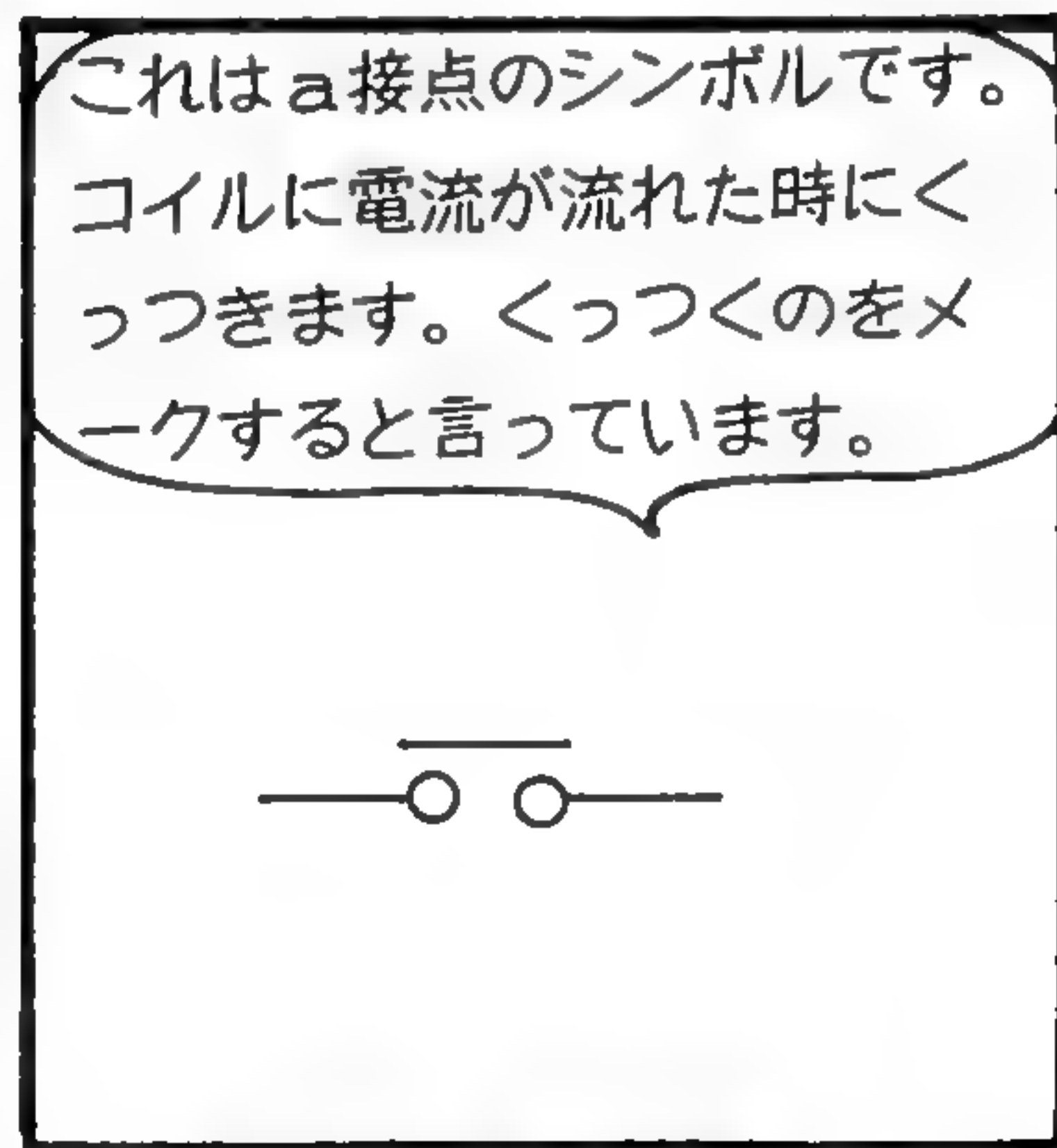
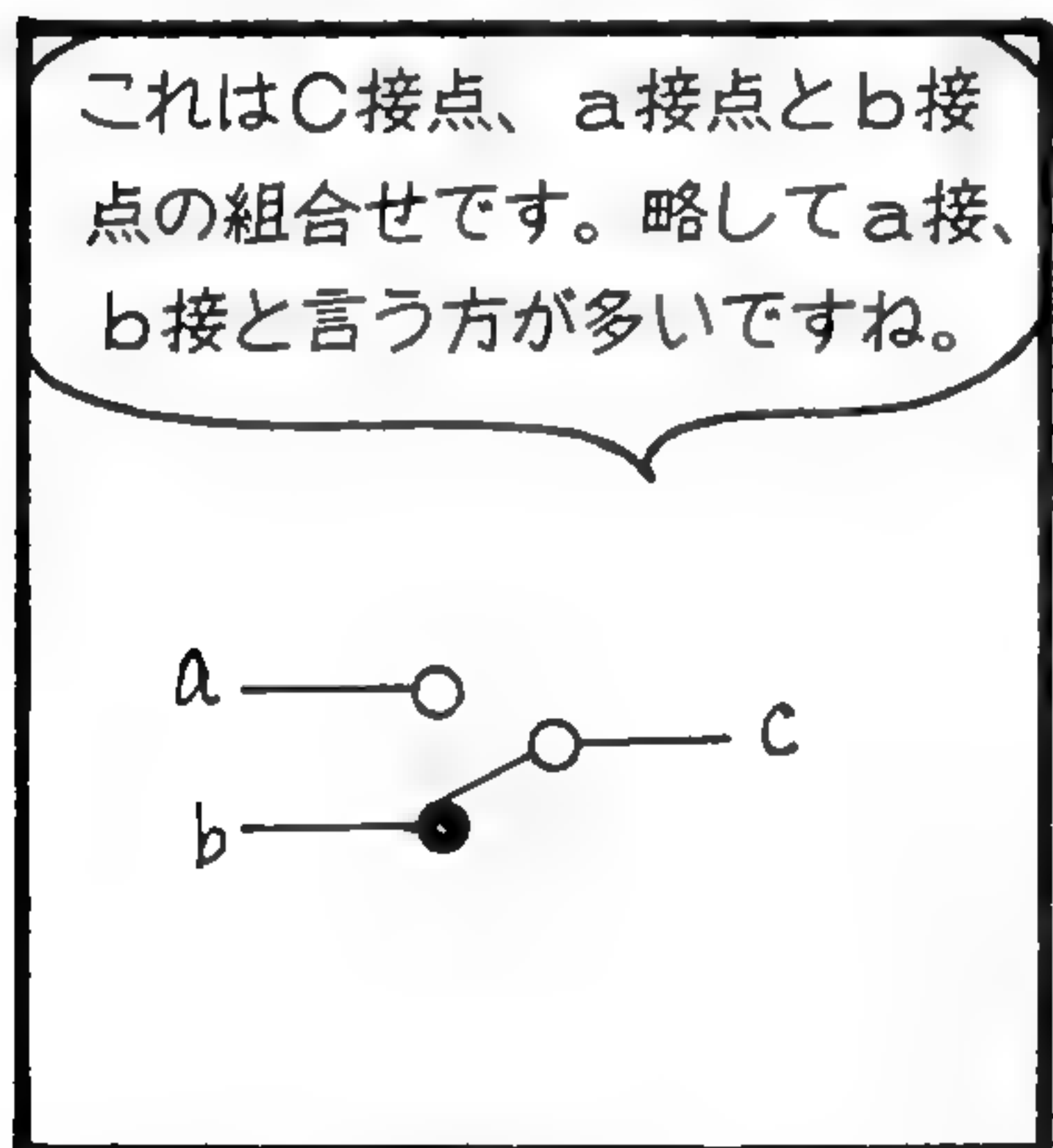
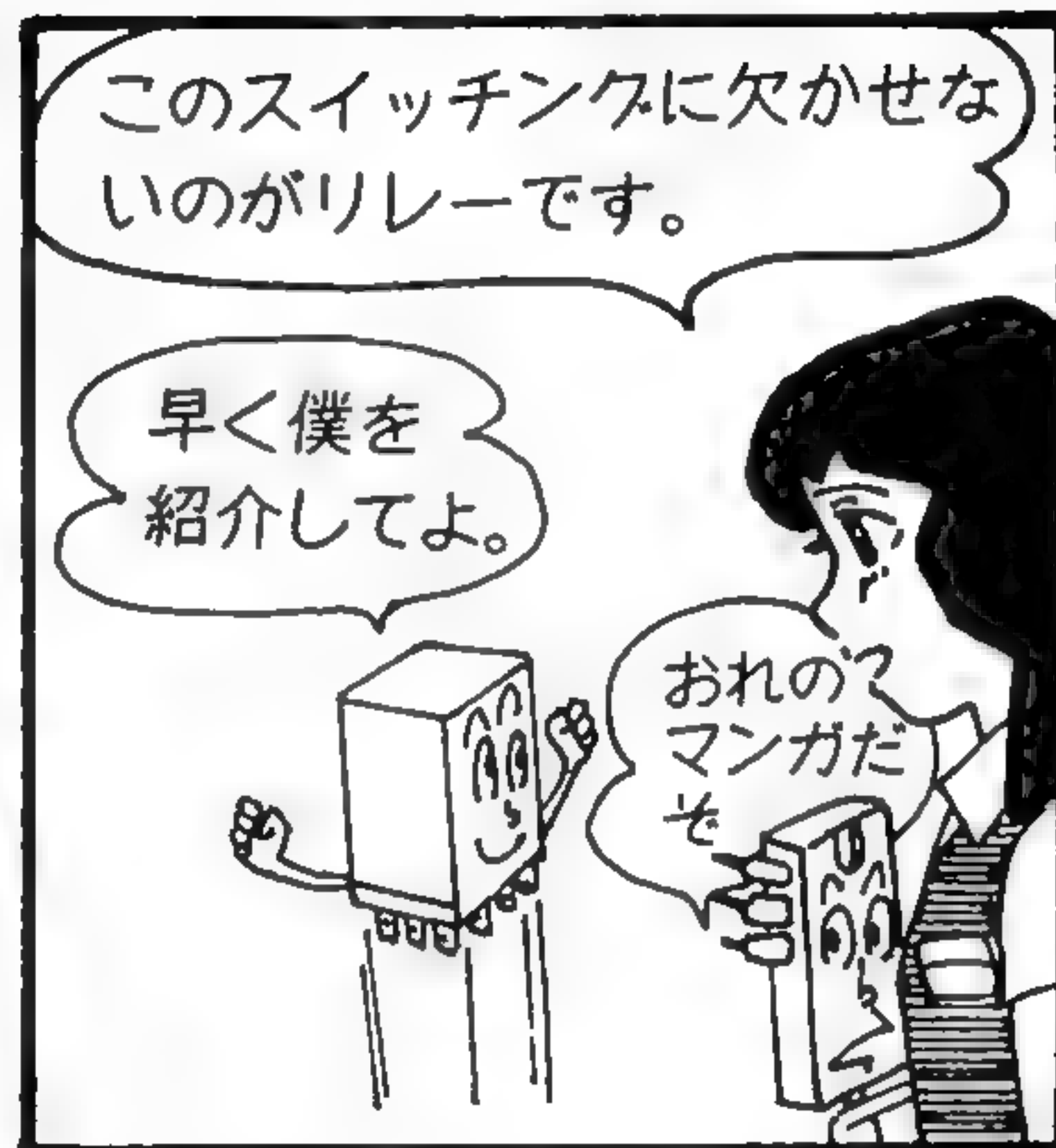
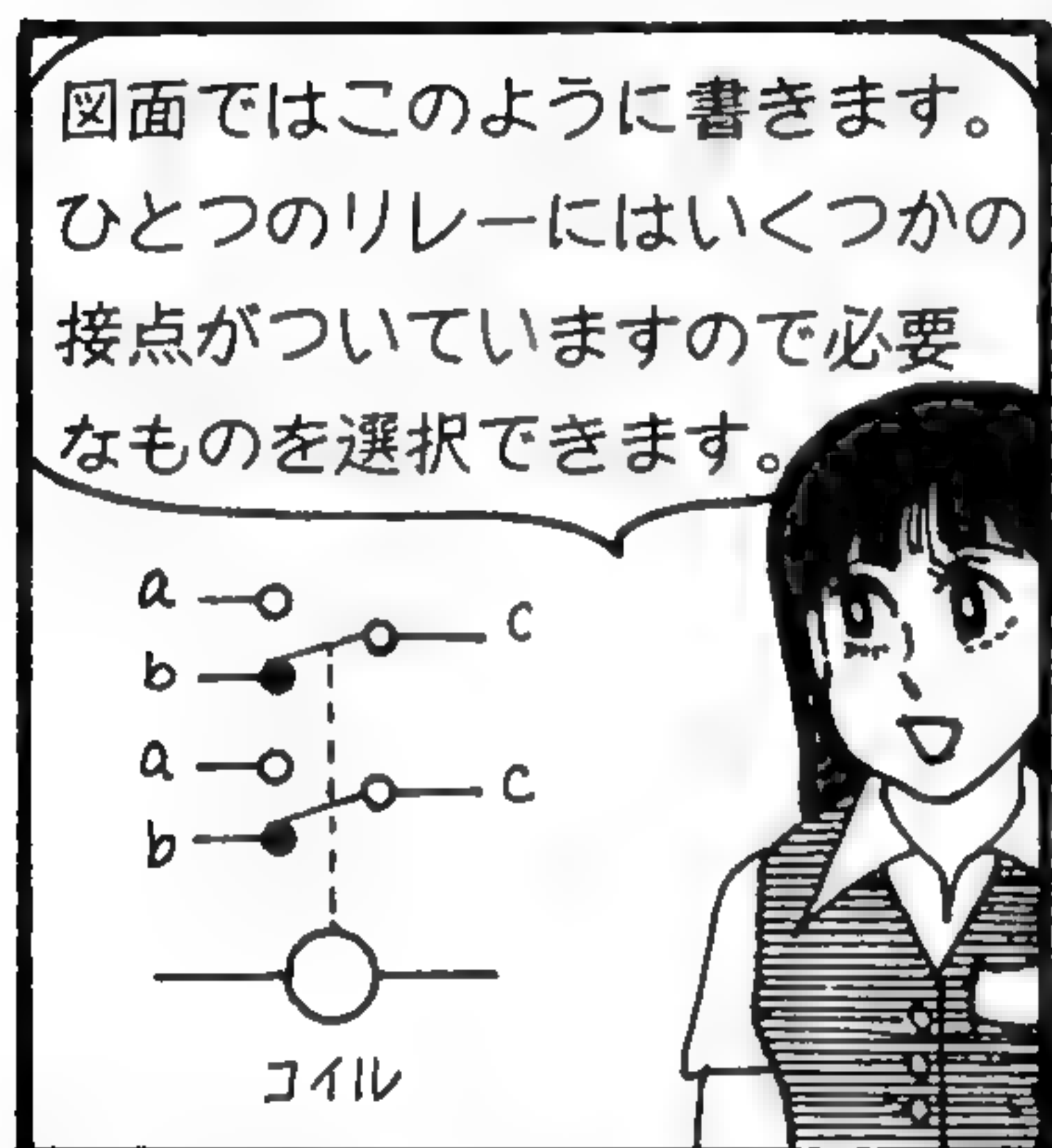
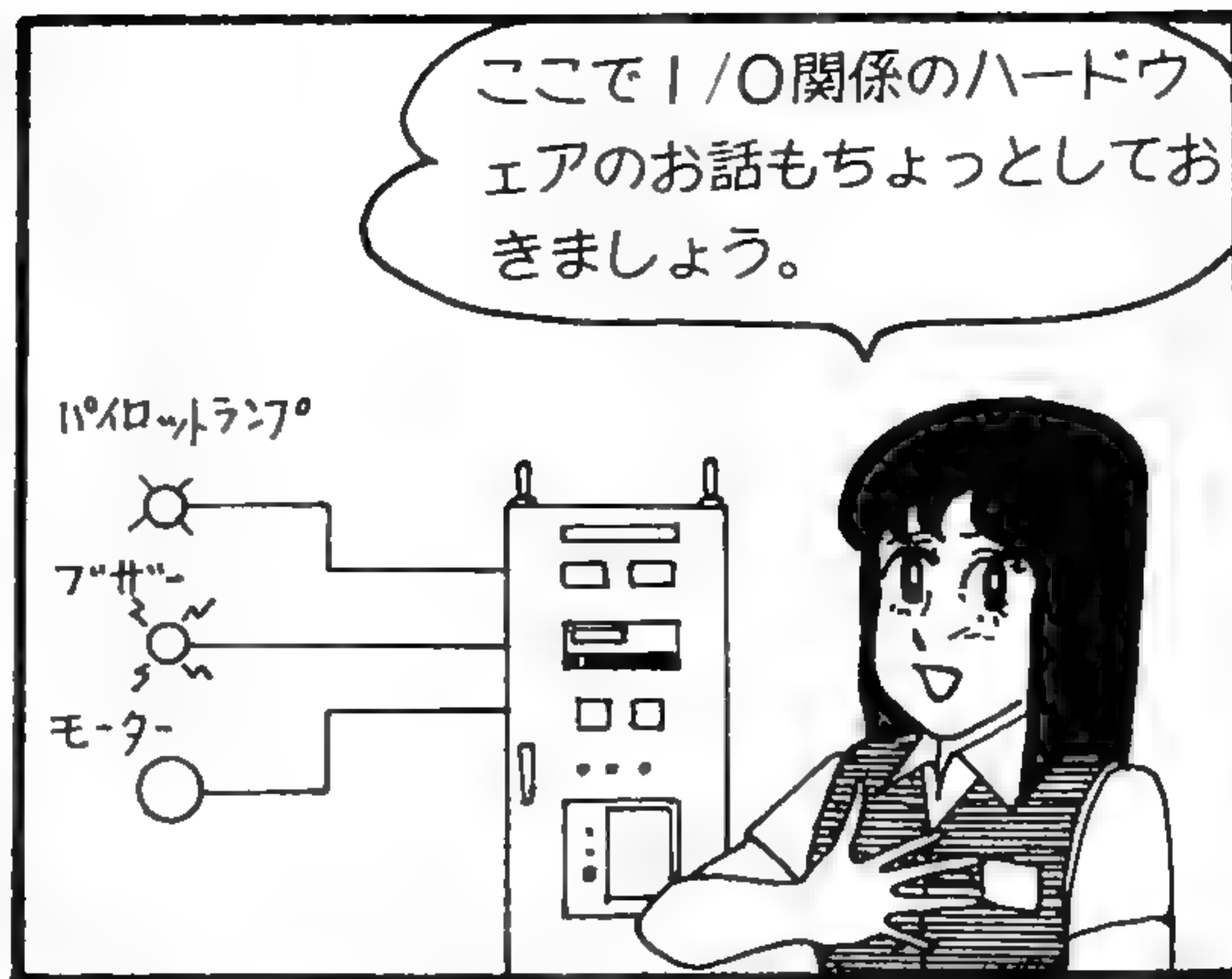
メーカー側では納入前にこのようなチェックを行なうのですがノイズ環境は設置場所によりそのつと違うというのが実情です。



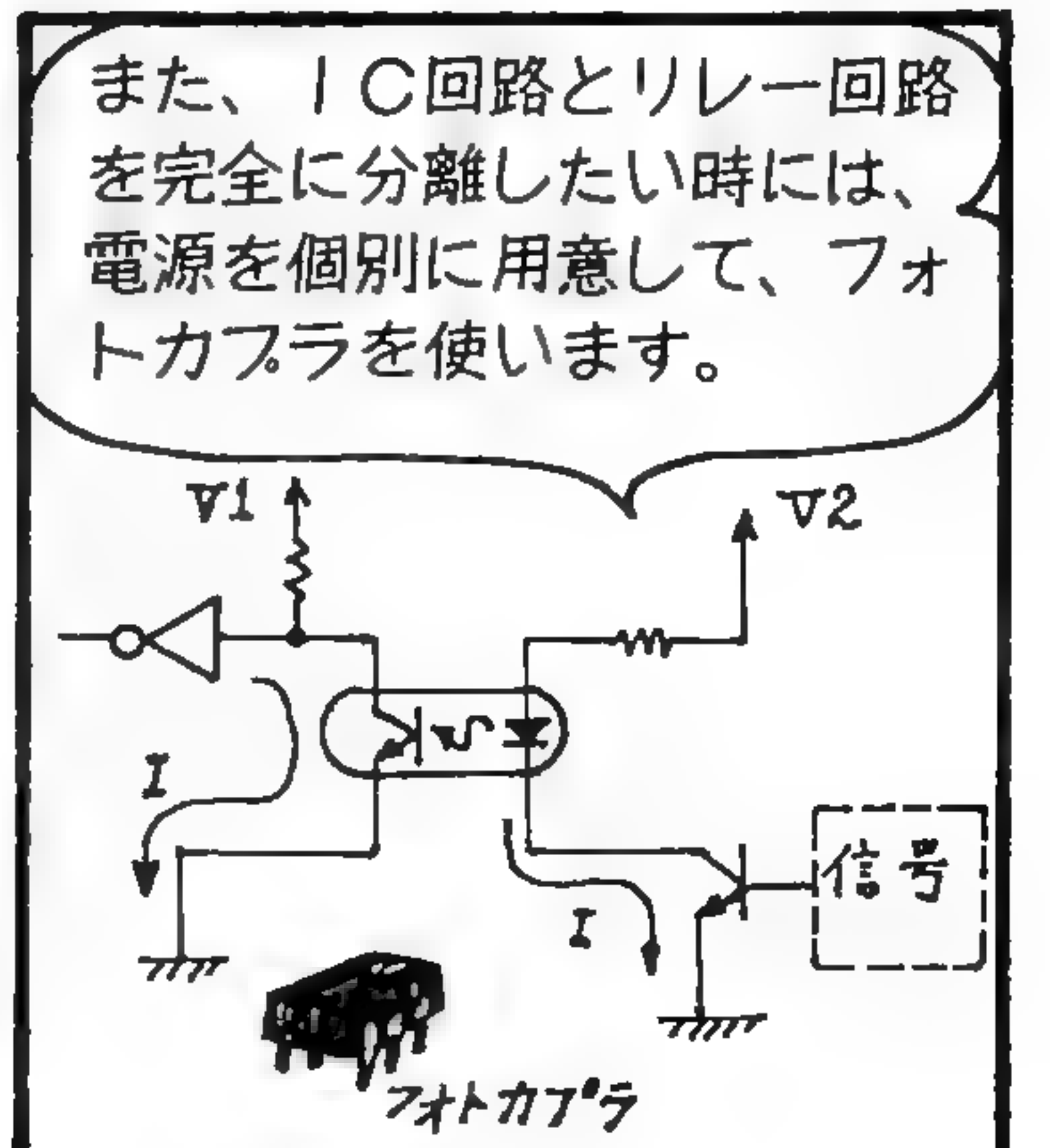
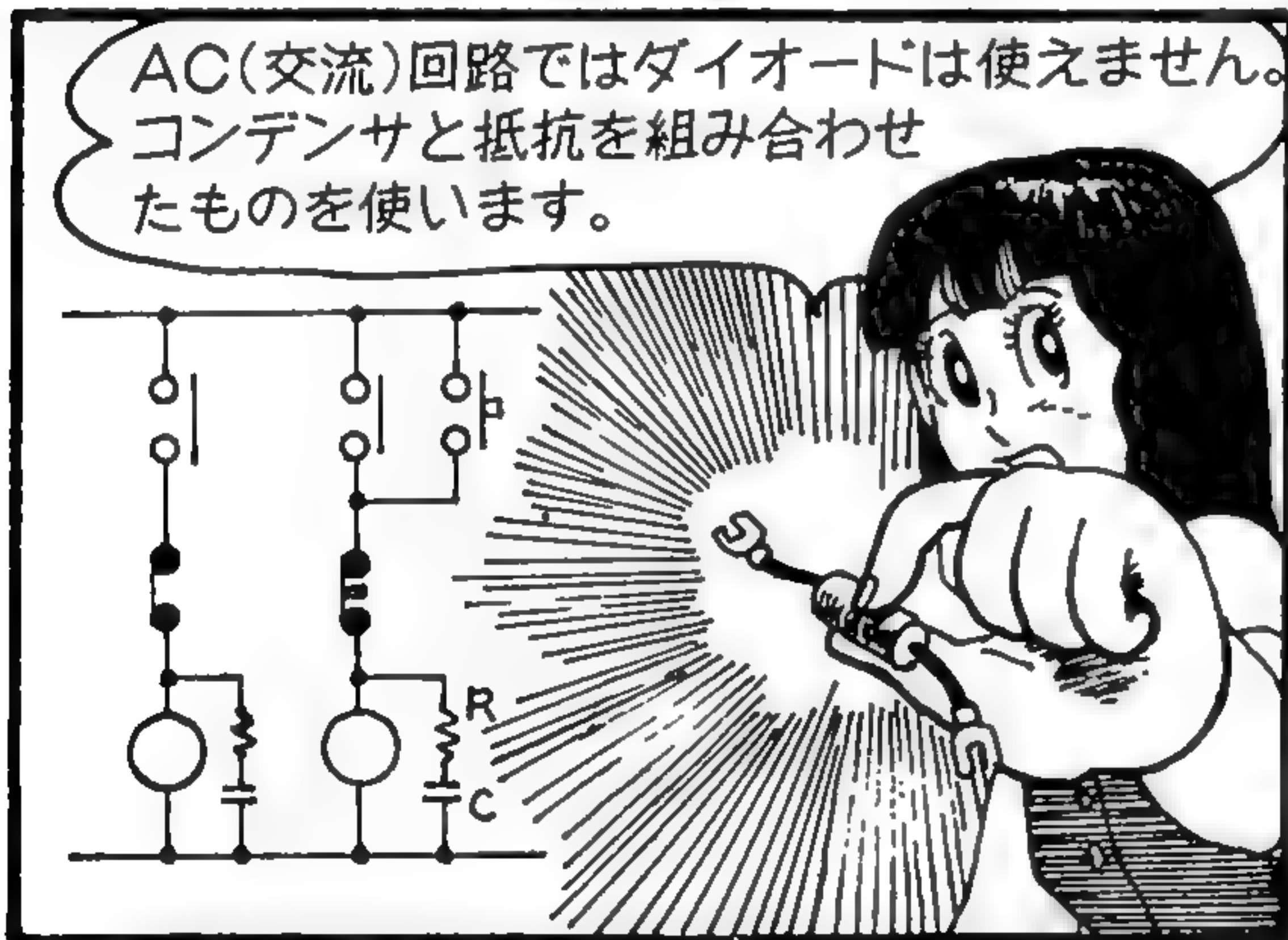
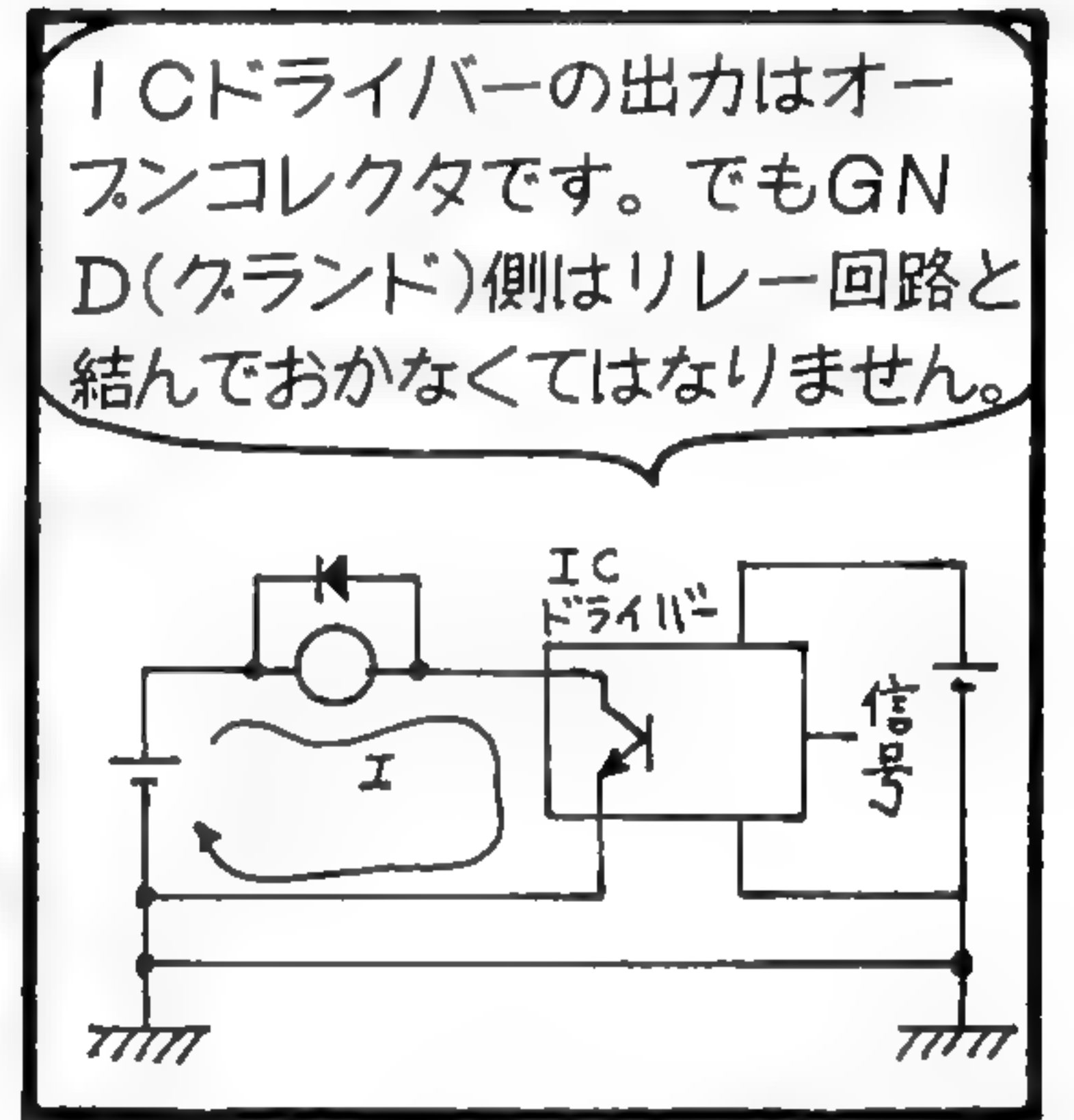
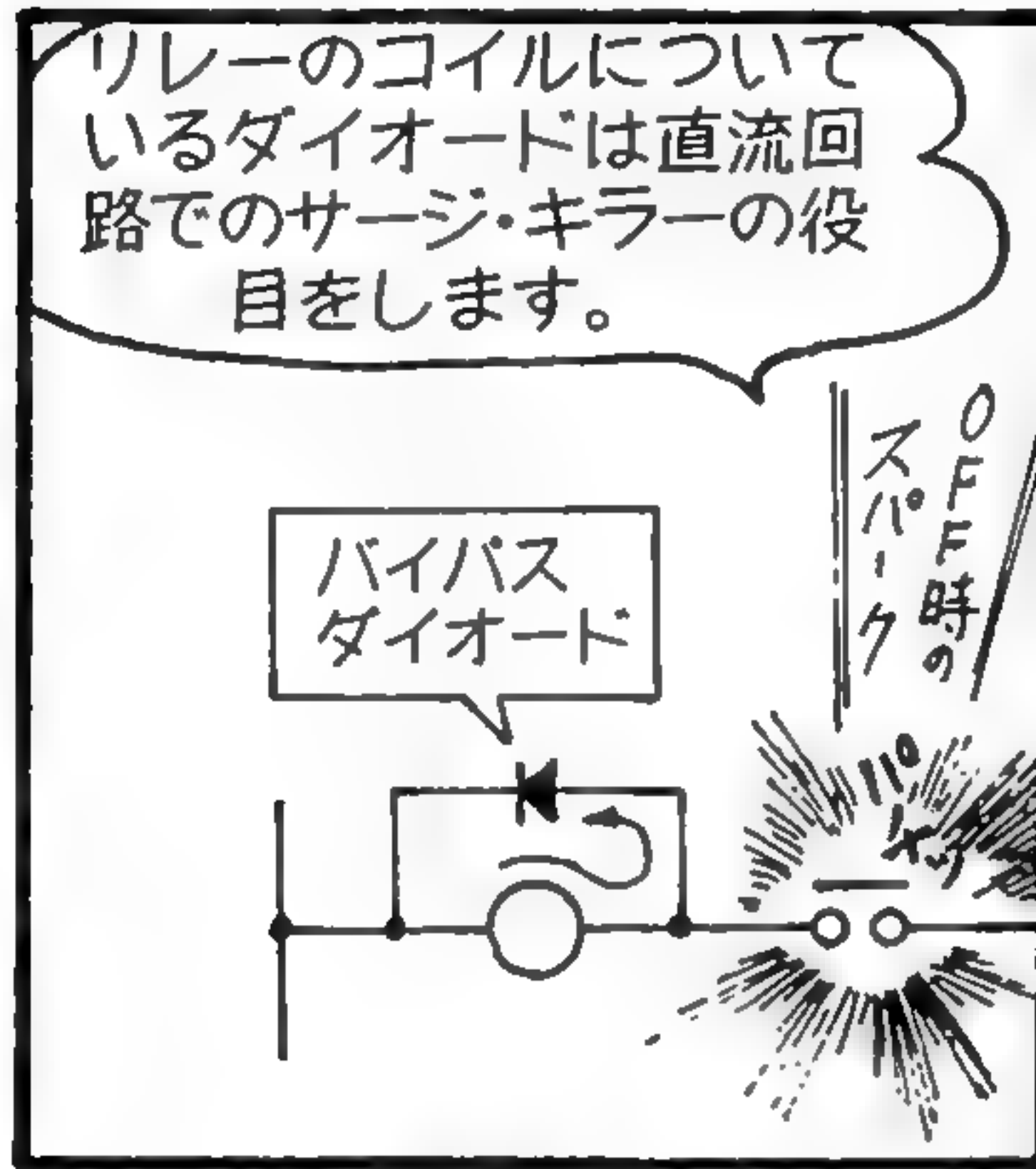
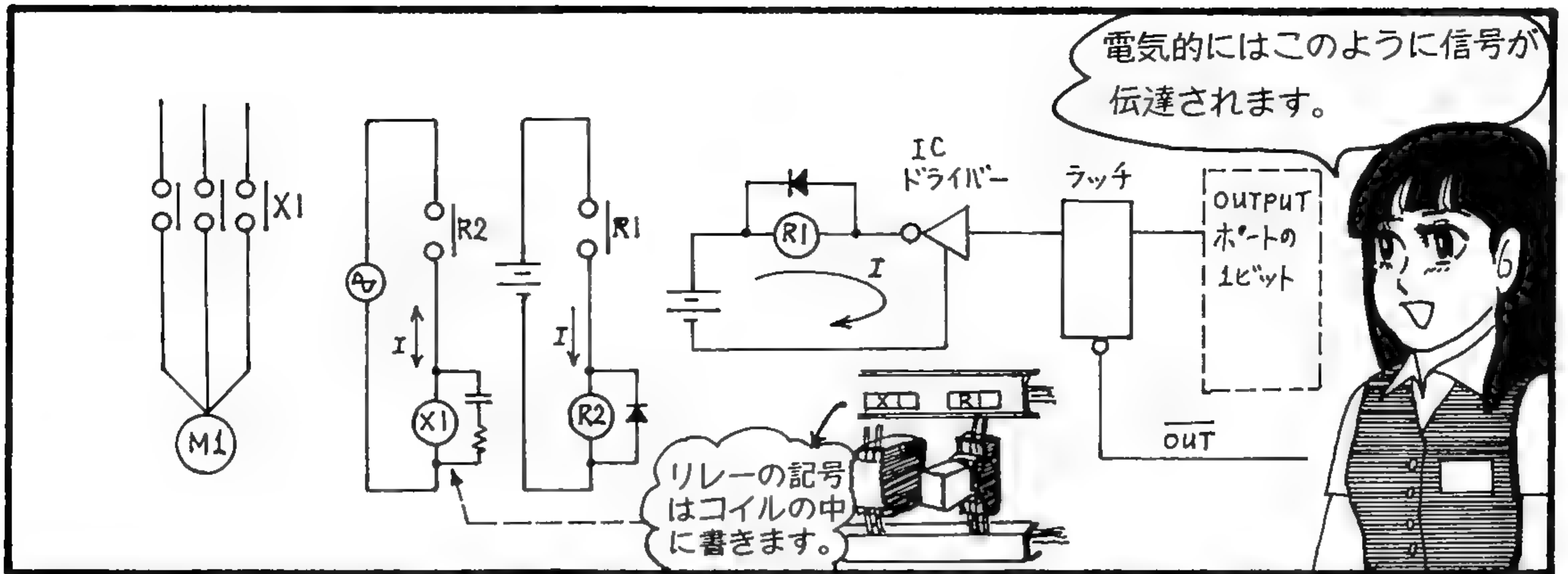
チェックはプリンタなどで一定時間間隔でプリントさせ、エージングを兼ねて行ないます。



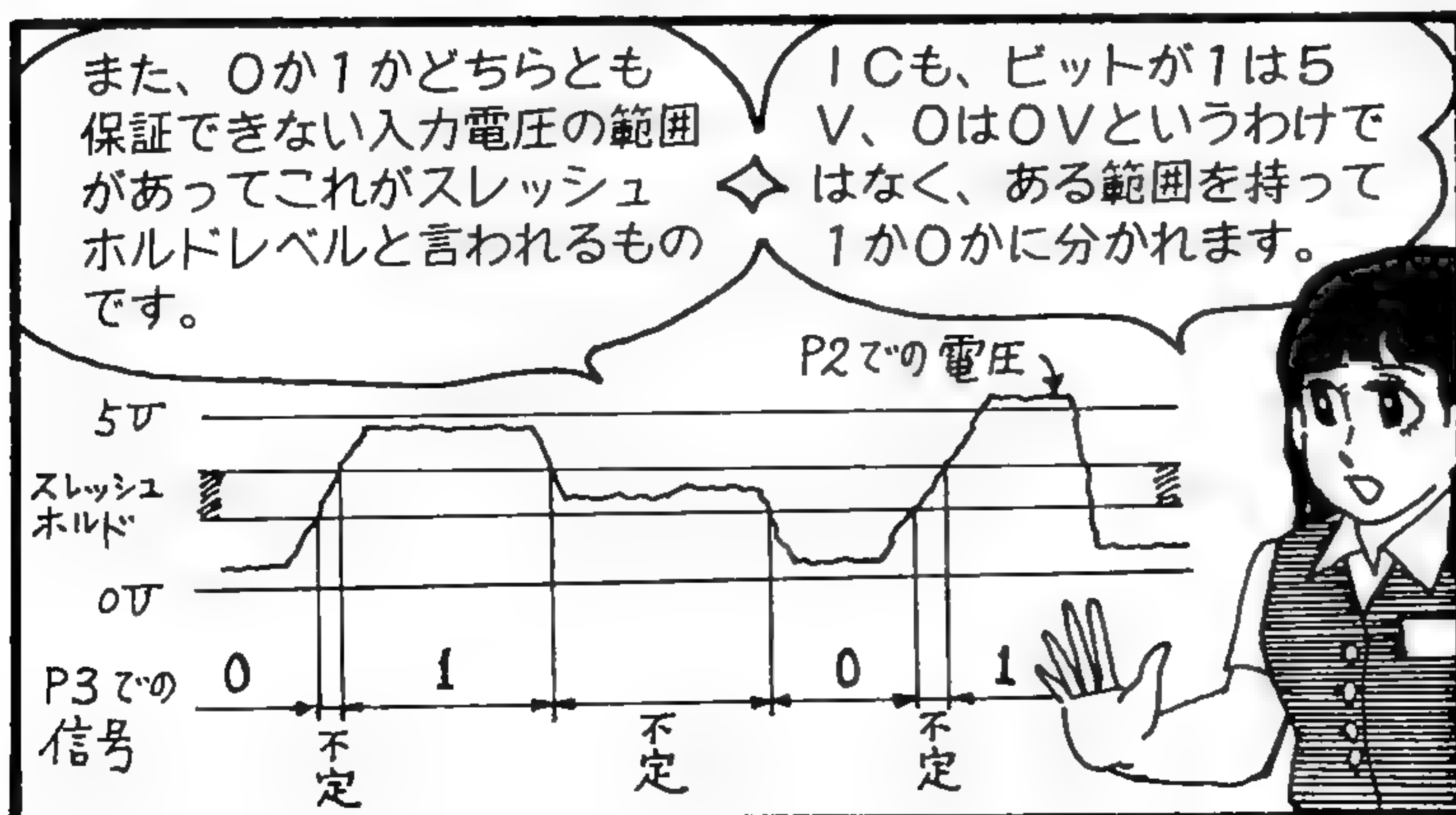
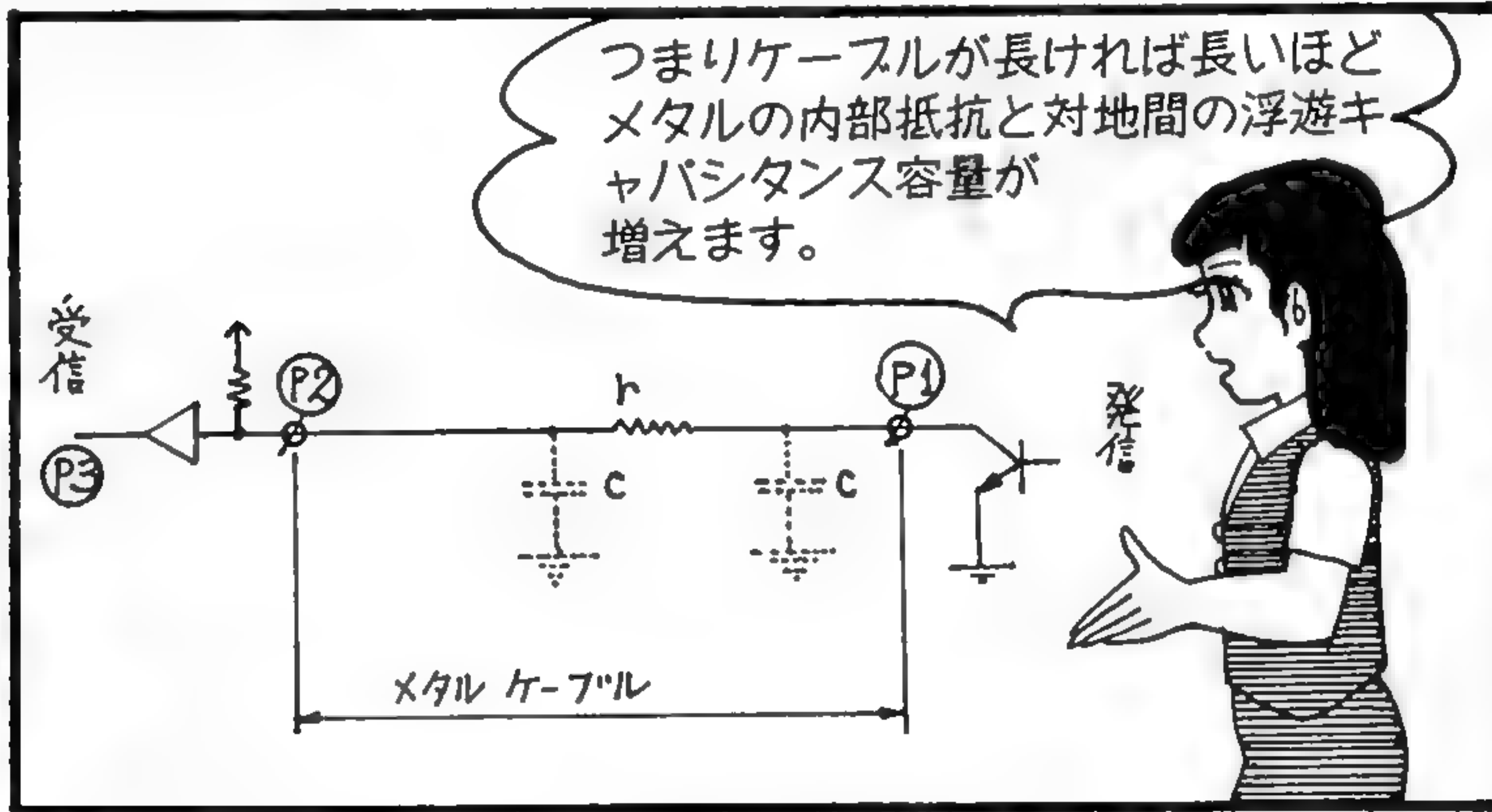
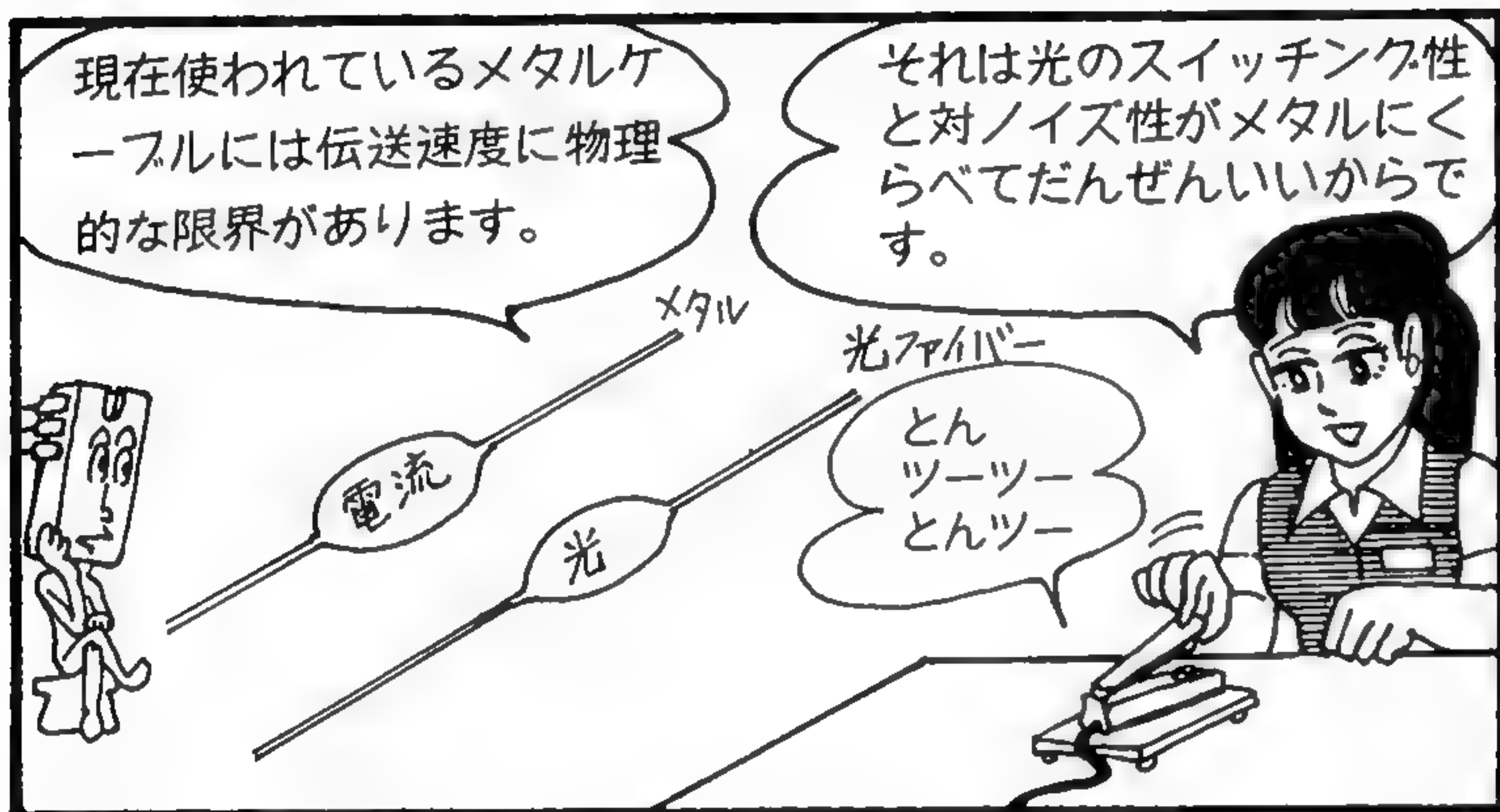
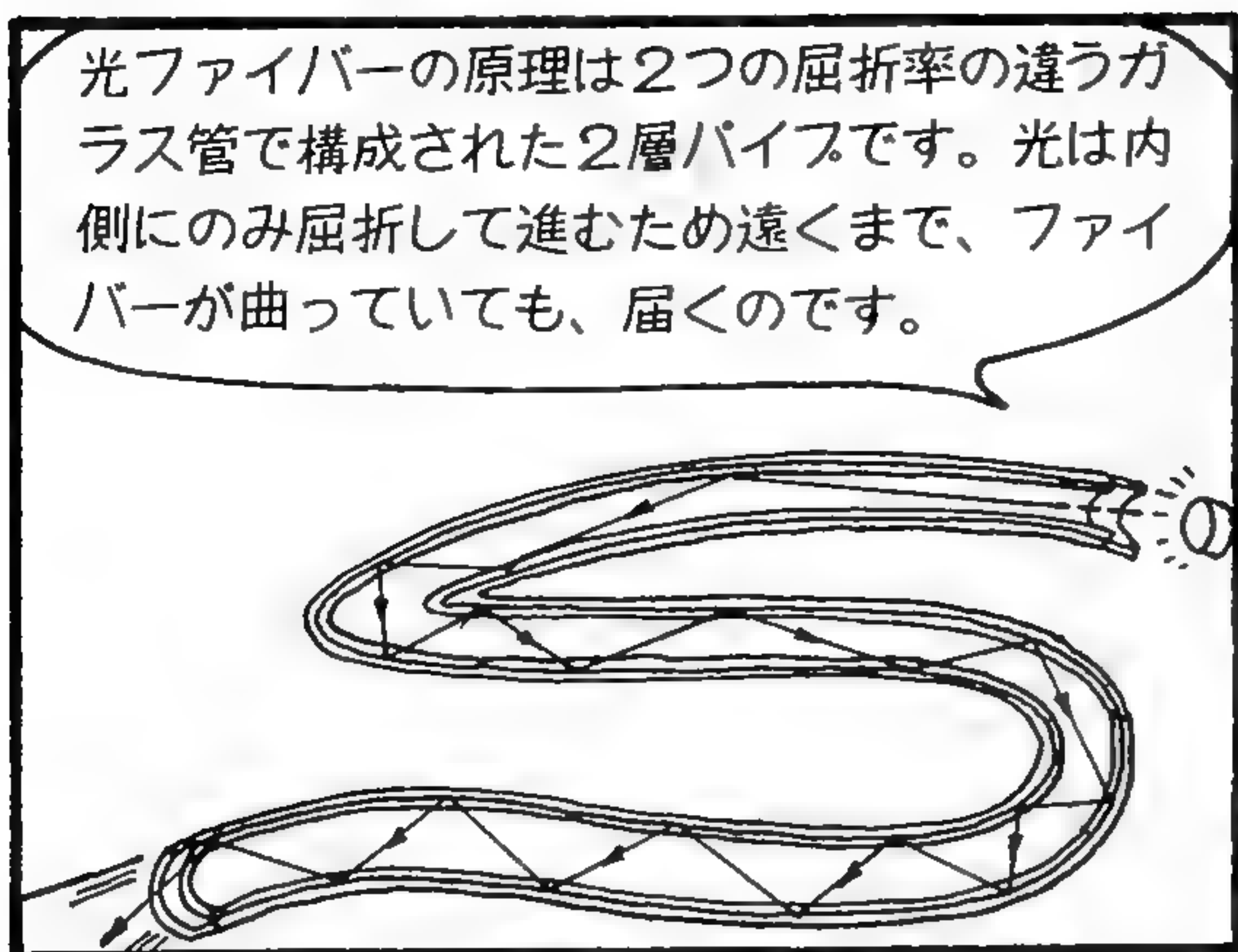














重要な信号線はシールドする必要があります。

という具合に途中で外部の磁界や電界の影響を受けやすいということです。

メタル通信にはもうひとつ決定的な弱点があります。それは…あら、いい男…

3つのパルスになっており発信内容とは違った情報が伝達されることになる。

通信回線のメタルに強力なサージ電圧が発生して、それにつながっているコンピュータをはじめとする電子機器がパンクする危険があります。

核爆発による強力な電磁波により、

たとえば電子化の進んだ都市が核攻撃を受けた場合など、

某国電子部品老朽からミサイルを誤射…

この点、光通信は光ったかどうかビット情報となるためファイバーが破壊されない限りOKなわけです。

うん 届いてる

直接破壊されなくても、情報機能の停止等により都市は大きなダメージを受けます。

こんなわけで、光ファイバーは注目を集めているのです。

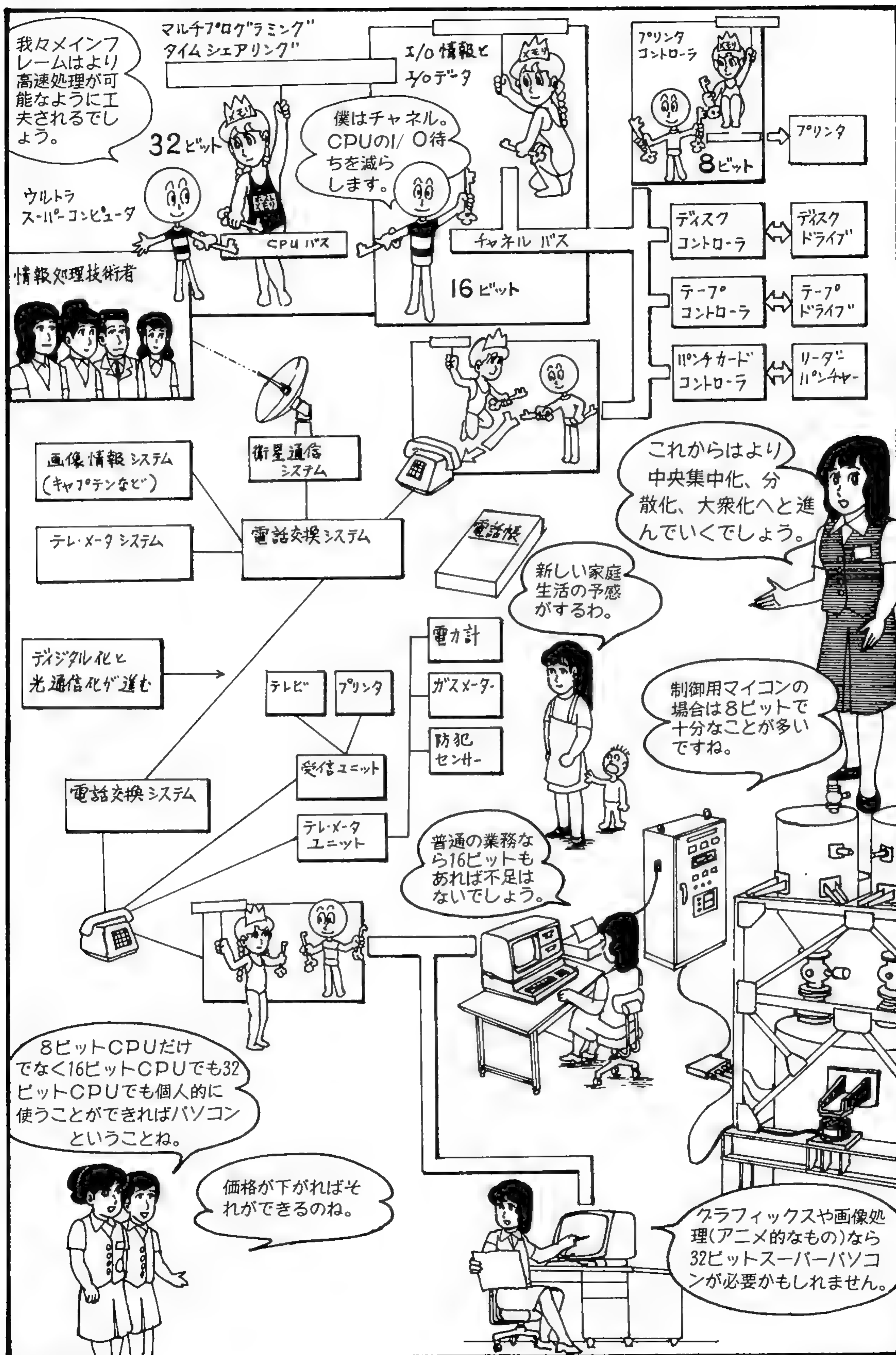
ファイバー

メタル

ファイバーの両端は発光ダイオードと受光ダイオードで対になっており、ファイバー自体が長い長いフォトカプラーと言えます。2つの回路を電氣的に分離することにもなります。

発信パルスと同じであり情報が正しく伝達される。







この本むずかしかった  
でしょうか？

あなたがもし、中学生や高校  
生ならそれが当たり前です。  
大学生でもスラスラ読めた人  
はそういないと思います。

マンガの書き方もあるかもし  
れませんが、なにしろ相手は  
小さいとはいえコンピュータ  
ですからね。むずかしいのが  
当たり前かもしれません。

ハードウェアの方は時々刻々  
と改良が加えられ新しいもの  
に置き変わっていきます。で  
すから移り変わっていく部分  
よりも本質的なことを理解す  
ることが大切だと思うのです。

時代が16ビット、32ビット  
に進みメモリの大容量化、  
小型化、スピードアップが  
進もうとも、ここに書いた  
基本的なことは、真空管の  
時代から変わっていないの  
です。

では、ひとまず  
これで失礼します。  
ごきげんよう、  
さようなら。





絵とき

8ビットマイコンCPU大研究 定価 1500 円

昭和59年9月10日 初版発行

著者 笹 倉 正 義

発行者 中 村 洋 一 郎

発行所 株式 日本実業出版社  
会社

東京都文京区本郷3丁目2番12号 ☎ 113

☎ 代表 03(814)5161 振替 東京 7-25349

大阪市北区西天満6丁目8番1号 ☎ 530

☎ 代表 06(362)6141

印刷所 壮光舎印刷株式会社

製本所 株式会社 若林製本工場

落丁、乱丁本はお取替え致します

©M.Sasakura 1984. Printed in JAPAN

ISBN4-534-00962-3 C0055 Y1500E



## はじめての人のマイコンがわかる本

矢矧晴一郎 定価1000円

マイコンを買おうという人、興味のある人に役立つ知識集。読み進めるうちにいつの間にか、マイコンのハード・ソフトから周辺機器の活用法までがわかり、メキメキ知識が増え、関心が広がる本。「知ったかぶり」をしたい人も利用できる。

## マイコンとマイコンをつなぐ法

藤原君恵・石塚日出子 古賀義亮監修 定価1100円

二台以上のマイコンをつないでデータのやりとりができるオンラインのつくり方を手とり足とり教える。線のつなぎ方、交信プログラムのつくり方、遠隔地でも交信できる音響カブラの使い方、いま話題のハンドヘルド・コンピュータ活用法等。

ビジネスマンのための

## マイコンBASIC入門

藤原君恵・石塚日出子 定価1100円

これから始める人、カベにぶつかっている人、何とかもう一度と思う人のためのBASIC入門。BASICはわかりにくいという伝説を信じている人もぜひ。BASICとはどういうもので、プログラムはどうつくればよいのかをやさしく解説。

ビジネス用

## マイコンBASICプログラムのつくり方

笠原耕文 定価1200円

「知識の時代から活用の時代に入った」といわれるマイコンを、自由自在に使いこなすための手引書。ディスプレイのテクニック、データ入・出力のためのプログラム、ファイルのつくり方・考え方、データファイルの処理プログラムなど。

ビジネス用

## マイコンにくわしくなる本

新版

岩崎俊雄 定価1100円

マイコンを知識の段階から一歩進め、実際の仕事に十分役立てるためのガイド。どんな事務処理にも応用できるデータ処理のプログラムのつくり方をくわしく述べ、読者の応用に役立つ。基礎からBASIC、ディスクファイルの活用まで。

## 誰にでもわかる マイコン入門

桑山義明 定価980円

“マイコン時代”というけれど、まだ見たこともさわったこともない——と内心不安に思っている人のためのマイコン入門の決定版。どんな機種があるか、どんな形をしていてどう操作するのか、どんなしくみで何ができるのか、BASICとは等々。

## 絵でわかるコンピュータ

石田晴久監修 定価980円

コンピュータについてまったくのシロウトでも、絵を見ているうちにコンピュータとはどのようなものかがわかる絶好の入門書。コンピュータで何ができるか、コンピュータによる計算法とそのしくみ、ハードウェア・ソフトウェアとは……等。

## SORD-PIPSを100%使いこなす法

西 順一郎編 定価1000円

普通の使い方からオフコン並みの活かし方まで、簡易言語のNo.1、PIPSのいろいろな活用のアイデアを満載。マイコン教室でも教えてくれない、マニュアルにも書いてない使いこなす方がマスターできる。BASICに絶望した人にもわかる。

コンピュータ

## ソフトウェアのことがわかる本

水野健児 定価1100円

「コンピュータ、ソフトがなければタダの箱」といわれるように、生かすも殺すもソフト次第。OS（オペレーティング・システム）からアプリケーション・プログラムまで、好き嫌い、得手不得手に関係なくソフトウェアがわかる入門書。

## コンピュータ用語の意味がわかる辞典

石田晴久監修 定価1300円

コンピュータの勉強でつまずくのが用語の意味。よく見聞きするコンピュータ用語約1500語をあらゆる分野から収録。専門用語を使った説明はできるだけ避け、「要するにこういう意味なんですよ」というふうにシロウトにも理解できる言葉で説明。

## ワープロ・パソコンのタイピング入門

山元祥弘 定価980円

ワープロ・パソコンを使いこなす第一の条件であるタイピングの打ち方を、初心者のために手とり足とり教える。誰でもがJISキーボードを早く正しく打てるようになる。自己流で、悪いクセのついてしまっている人も、これで基本をマスターできる。

## ICとLSIのことがわかる本

久野英雄 定価1100円

コンピュータから家庭電化製品まで、身の回りのあらゆるものを小型化・高性能化・低価格化しているICとLSI。そのしくみから産業最前線までのすべてを、文科系の人にもわかるよう平易に解説。現代ビジネスマン必須の知識・教養が身につく。







ISBN4-534-00962-3 C0055 ¥1500E

定価 1500円